VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR STŘÍDAVÝCH ELEKTRONICKÝCH OBVODŮ

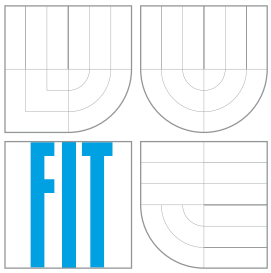
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN JANKO

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

SIMULÁTOR STŘÍDAVÝCH ELEKTRONICKÝCH OBVODŮ

AC ELECTRONIC CIRCUITS SIMULATOR

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

ROMAN JANKO

VEDOUCÍ PRÁCE
SUPERVISOR

doc. Ing. JIŘÍ KUNOVSKÝ, CSc.

BRNO 2011

Abstrakt

V práci jsou stručně uvedena vybraná numerická řešení lineárních diferenciálních rovnic a některé standardní metody vyšetřování frekvenčních a fázových charakteristik elektronických obvodů. Obsahem práce je také malá sbírka ukázkových příkladů pro vyšetřování frekvenčních a fázových charakteristik elektronických obvodů R, L, C. Je zde navrhnuto a implementováno uživatelské rozhraní pro schematické zadávání elektronických obvodů a pro grafické zobrazení frekvenčních a fázových charakteristik zkoumaného elektronického obvodu. V závěru práce je stručně zhodnocena časová náročnost výpočtu vybraných frekvenčních a fázových charakteristik elektronických obvodů.

Abstract

The thesis summarizes the selected numerical solutions of linear differential equations and some standard methods for investigating the frequency and phase characteristics of electronic circuits. The part of thesis is also a small collection of sample examples to investigate the frequency and phase characteristics of R, L, C electronic circuits. There are designed and implemented user interface for electronic circuit schematic entry and graphical display of frequency and phase characteristics of the analysis of electronic circuits. In conclusion, this thesis is briefly reviewed time-consuming calculation of selected frequency and phase characteristics of electronic circuits.

Klíčová slova

Simulátor, charakteristiky elektronických obvodů, frekvenční a fázové charakteristiky, diferenciální rovnice, SIMLIB, TKSL.

Keywords

Simulator, characteristics of electronic circuits, frequency and phase characteristics, differential equations, SIMLIB, TKSL.

Citace

Roman Janko: Simulátor střídavých elektronických obvodů, bakalářská práce, Brno, FIT VUT v Brně, 2011

Simulátor střídavých elektronických obvodů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením doc. Ing. Jiřího Kunovského, CSc. a uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Roman Janko
9. května 2011

Poděkování

Děkuji doc. Ing. Jiřímu Kunovskému, CSc. za jeho odborné vedení, podnětné připomínky a rady, směrování práce a za poskytnutí vynikajících informačních zdrojů moderního systémového řešení simulačních modelů.

Obsah

1 Úvod **4**

2 Vybrané elektrické obvody **5**
2.1 Účel a použití filtrů . . . 5
2.2 Rozdělení filtrů . . . 5
2.3 Frekvenční charakteristiky . . . 6

3 Metody řešení střídavých elektrických obvodů **8**
3.1 Fázorové diagramy . . . 8
3.2 Symbolicko-komplexní metoda . . . 10
3.3 Diferenciální rovnice . . . 11

4 Metody pro numerické řešení lineárních diferenciálních rovnic **12**
4.1 Jednokrokové metody . . . 13
4.1.1 Eulerova metoda . . . 13
4.1.2 Metoda Runge-Kutta . . . 15
4.2 Vícekrokové metody . . . 16

5 Vyšetřování frekvenčních charakteristik a jejich konstrukce **17**
5.1 Amplitudová charakteristika . . . 19
5.2 Fázová charakteristika . . . 19
5.3 Vyšetřování charakteristik na počítači . . . 20

6 Popis řešení simulátoru střídavých elektronických obvodů **22**
6.1 Ovládání programu . . . 22
6.2 Pomocné knihovny . . . 25
6.2.1 Qt . . . 25
6.2.2 TKSL . . . 26
6.2.3 SIMLIB . . . 27
6.3 Vývojový diagram činnosti programu . . . 29
6.4 Implementace vytvořeného programu . . . 31
6.4.1 Třída MainWindow . . . 31
6.4.2 Třída Editor . . . 32
6.4.3 Třída Soucastka . . . 34
6.4.4 Třída Spoj . . . 34
6.4.5 Třída Graf . . . 34
6.4.6 Třída Simulace . . . 34

7 Hodnocení algoritmu řešení vytvořeného simulátoru **39**
7.1 Časová náročnost výpočtu . . . 39
7.2 Srovnání výsledků se světovými standardy . . . 39

8 Příklady pro vyšetřování frekvenčních charakteristik obvodů R, L, C **41**
8.1 Příklad 1 . . . 41
8.2 Příklad 2 . . . 42
8.3 Příklad 3 . . . 43
8.4 Příklad 4 . . . 44
8.5 Příklad 5 . . . 45
8.6 Příklad 6 . . . 46
8.7 Příklad 7 . . . 47
8.8 Příklad 8 . . . 48

9 Závěr **50**

Literatura **51**

A Metriky kódu **52**

B Hlavičkové soubory naimplementovaných tříd **53**
B.1 Třída MainWindow . . . 53
B.2 Třída Editor . . . 55
B.3 Třída Soucastka . . . 57
B.4 Třída Spoj . . . 58
B.5 Třída Graf . . . 60
B.6 Třída Simulace . . . 62

Seznam obrázků

2.1 Amplitudová charakteristika pro horní propust HP 6
2.2 Fázová charakteristika pro horní propust HP 7

3.1 Časový průběh napětí 8
3.2 Schéma sériového RLC obvodu 9
3.3 Fázorový diagram sériového RLC obvodu 9

4.1 Přesné a přibližné řešení diferenciální rovnice 12
4.2 Odvození Eulerovy metody 13
4.3 Srovnání numerického a analytického řešení 15

5.1 Zobrazení průběhů napětí na osciloskopu 17
5.2 Ustálený harmonický signál 18
5.3 Neustálený signál 19
5.4 Možnosti odečtení fázového posunu 20
5.5 Problém lokalizace maxima 21

6.1 Hlavní okno programu s editorem pro schématické zadávání el. obvodů . . . 22
6.2 Dialogové okno pro zadání parametrů simulace 23
6.3 Průběh vyšetřování charakteristik 23
6.4 Zobrazení amplitudové frekvenční charakteristiky 24
6.5 Zobrazení fázové frekvenční charakteristiky 24
6.6 Signály a sloty v Qt 26
6.7 Vývojový diagram běhu programu 30
6.8 Dialogové okno pro nastavení simulace 30
6.9 Výpis parametrů simulace a možnost výběru mezi charakteristikami 32

7.1 Charakteristiky filtru pomocí systému Matlab 40

8.1 Integrační článek - dolní propust 41
8.2 Derivační článek - horní propust 42
8.3 Wienův článek - pásmová propust 43
8.4 Rezonanční pásmová propust 44
8.5 Rezonanční pásmová zádrž 45
8.6 Sério-paralelní RLC článek 46
8.7 Pásmová propust 47
8.8 Čebyševův filtr 48

Kapitola 1

Úvod

Simulace na číslicových počítačích je velice rozšířená disciplína. Uplatnění nachází ve velkém množství oborů jako medicína, fyzika, chemie, astronomie, meteorologie, ekonomika, technika a další. Výhodou simulačních metod je jejich cena, rychlost a bezpečnost. Existují dva odlišné přístupy k simulaci: diskrétní a spojitý.

Práce je zaměřena na spojitou simulaci, pro elektrické obvody. Existuje několik způsobů řešení elektronických obvodů. V textu práce se budu věnovat především jejich řešení pomocí diferenciálních rovnic. K řešení diferenciálních rovnice existuje množství metod jako Eulerova metoda a metoda Rungeho-Kutta.

Cílem práce je implementace simulátoru střídavých elektronických obvodů. Konkrétně R, L, C elektronických obvodů tzv. pasivních kmitočtových filtrů. Tyto obvody jsou frekvenčně závislé. Výstupem simulátoru jsou frekvenční charakteristiky - amplitudová a fázová charakteristika. Při vyšetřování frekvenčních charakteristik je nutné pro každou frekvenci znovu provést výpočet soustavy diferenciálních rovnic, odečíst potřebné hodnoty, vypočítat přenos a fázi a výsledky zakreslit do grafů. Navržený simulátor tento proces plně automatizuje.

Text práce je členěn do devíti kapitol. První čtyři kapitoly se věnují popisu vybraných elektrických obvodů, způsobům jejich řešení (zejména pomocí diferenciálních rovnic), metodám pro výpočet těchto rovnic a konstrukci výsledných frekvenčních charakteristik. Další kapitola se zabývá implementací simulátoru. Následuje hodnocení algoritmu zajišťující výpočet frekvenčních charakteristik a srovnání řešení se světovými standardy v oboru jako Matlab, Maple a Dymola. Poslední kapitola obsahuje vzorové příklady R, L, C obvodů a jejich popis pomocí diferenciálních rovnic.

Kapitola 2

Vybrané elektrické obvody

Kmitočtové filtry jsou elektrické obvody, které se vyznačují tím, že jsou frekvenčně závislé. To znamená, že velikost amplitudy výstupního napětí je závislá na frekvenci napětí vstupního.

2.1 Účel a použití filtrů

Filtry jsou dvojbrany, které propouští vstupní signál v určitém pásmu kmitočtů. Kmitočtové pásmo, ve kterém filtr propouští vstupní signál na svůj výstup, nazýváme *propustné pásmo*. Podle konstrukce filtru muže být signál v propustném pásmu i zesilován. Mimo propustné pásmo je vstupní signál naopak hodně utlumován - jedná se o tzv. *nepropustné pásmo*.

Kmitočtové filtry jsou součástí řady obvodů a systémů. Například dolní propust se používá v usměrňovačích, kde je třeba oddělit stejnosměrnou složku a potlačit všechny střídavé složky. Pásmová propust má např. uplatnění v přijímačích, kde vybírá signál určitého vysílače. Řadu příkladů lze uvést i z měřící nebo regulační techniky. Poslední dobou se v souvislosti s novými mikroelektronickými technologiemi (novými aktivními prvky a funkčními bloky) rozvíjí aktivní filtry RC. Induktory v RLC filtrech jsou totiž nejobjemnější, nejdražší a hlavně těžko integrovatelné součástky. Snažíme se je proto s filtrů odstranit resp. nahradit syntetickými ekvivalenty. [3]

2.2 Rozdělení filtrů

Můžeme rozeznat čtyři základní typy filtrů:

- dolní propust DP,
- horní propust HP,
- pásmovou propust PP a
- pásmovou zádrž PZ.

Vlastnosti filtrů dostatečně vyplývají z jejich názvu. Dále je možné rozdělit filtry do další skupiny:

- pasivní a
- aktivní filtry.

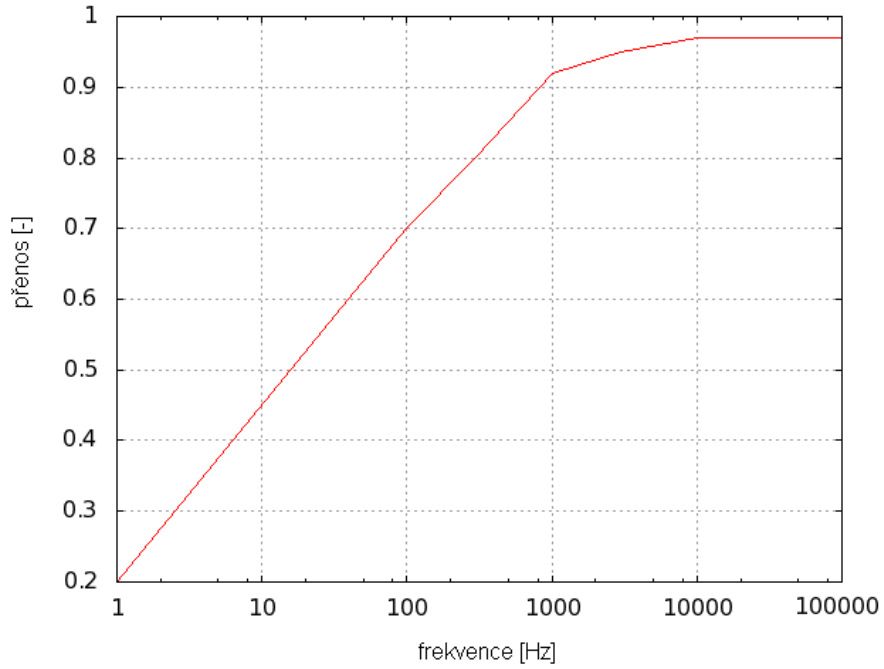
Pasivní filtry se vytvářejí pomocí rezistorů, kondenzátorů a cívek. Aktivní filtry obsahují navíc tranzistory nebo operační zesilovače. Aktivními filtry se označují zesilovače se záměrně kmitočtově závislým přenosem. Kmitočtová závislost zesílení se nejčastěji dosahuje pomocí vhodné zpětné vazby. Na rozdíl od pasivních filtrů lze dosáhnout mnohem větších přenosů v propustném pásmu.

Aktivní filtry přinášejí další výhodu, kterou je možnost realizace elektronické indukčnost vhodným zapojením zesilovače a kondenzátoru ve zpětné vazbě. I když je cívka je z teoretického hlediska velmi vhodná součástka pro konstrukci filtrů, její praktické použití je ovšem obtížnější. A to jednak z rozměrových a cenových důvodů a také z horší reprodukovatelnosti prvků se žádanou hodnotou indukčnosti. Často se uplatňují i nežádoucí, parazitní vlastnosti cívek, jako je činný odpor vinutí a parazitní kapacita. [6]

2.3 Frekvenční charakteristiky

Frekvenční charakteristiky slouží k popisu chování filtru v závislosti na frekvenci vstupního signálu.

- amplitudová charakteristika = zobrazuje závislost přenosu na frekvenci. Přenos se vypočítá jako poměr výstupní a vstupního napětí.

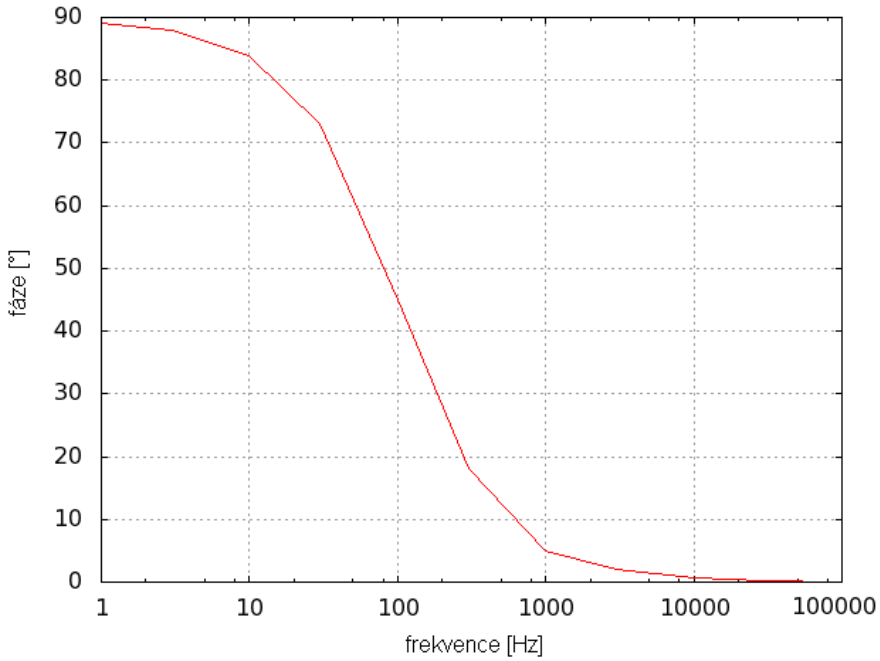


Obrázek 2.1: Amplitudová charakteristika pro horní propust HP
Zdroj: Vlastní

Amplitudová charakteristika pro horní propust je na obrázku 2.1. Je vidět, že filtr má nastaveno nepropustné pásmo do frekvence $100Hz$. Od této frekvence již propouští vstupní signál na výstup. Frekvence na pomezí mezi nepropustným a propustným pásmem (platí to samozřejmě i obráceně) je označována jako tzv. *mezní frekvence*. Mezní frekvenci filtru lze změnit nastavením jeho součástek. Možná obvodová realizace

filtru jako horní propusti je na obrázku 8.2. Zvýšíme-li například odpor rezistoru R, tak mezní frekvence klesne.

- fázová charakteristika = zobrazuje závislost fázového posunu mezi vstupním a výstupním signálem na frekvenci. Fázová charakteristika pro stejnou horní propust jako v minulém příkladu je na obrázku 2.2.



Obrázek 2.2: Fázová charakteristika pro horní propust HP
Zdroj: Vlastní

- fázorová charakteristika = zobrazuje závislost fázového posunu a přenosu na frekvenci v jednom grafu.

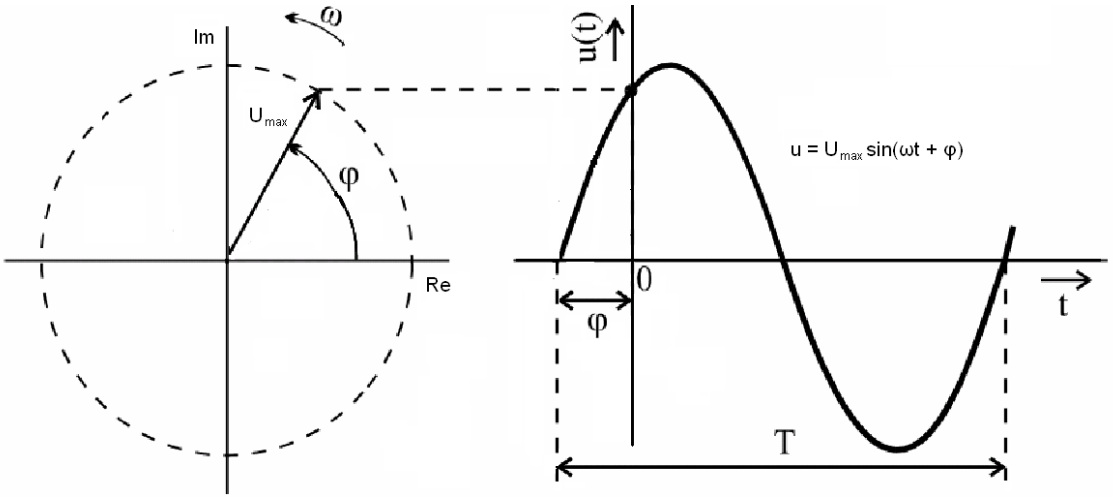
Metodám pro vyšetřování frekvenčních charakteristik a jejich konstrukci je věnována kapitola 5.

Kapitola 3

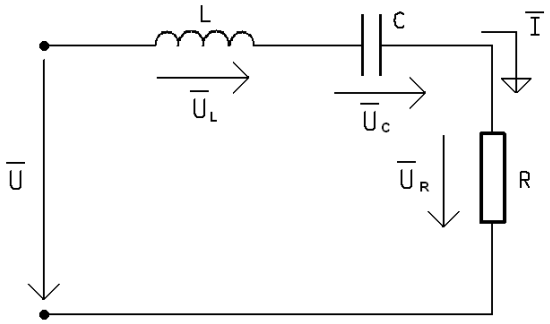
Metody řešení střídavých elektrických obvodů

3.1 Fázorové diagramy

Střídavé harmonické veličiny lze vyjádřit nejen pomocí goniometrických funkcích, ale také pomocí tzv. *fázorů*. Jak můžeme na obrázku 3.1 vidět, fázor je rotující úsečka v rovině a jednoznačně určuje danou sinusovou veličinu. Velikost fázoru (někdy také označováno jako modul) definuje velikost amplitudy střídavého signálu. Úhel mezi osou x a fázorem určuje fázi signálu. Fázory se zakreslují do fázorových diagramů. Jednotlivé fázory můžeme skládat. Skládat lze ovšem pouze fázory stejného druhu, tedy sčítat spolu jen proudy nebo spolu jen napětí. Použití fázorových diagramů není vhodné pro řešení složitějších obvodů. Diagram se pak stává nepřehledný. Následující příklad demonstruje počítání s fázory.



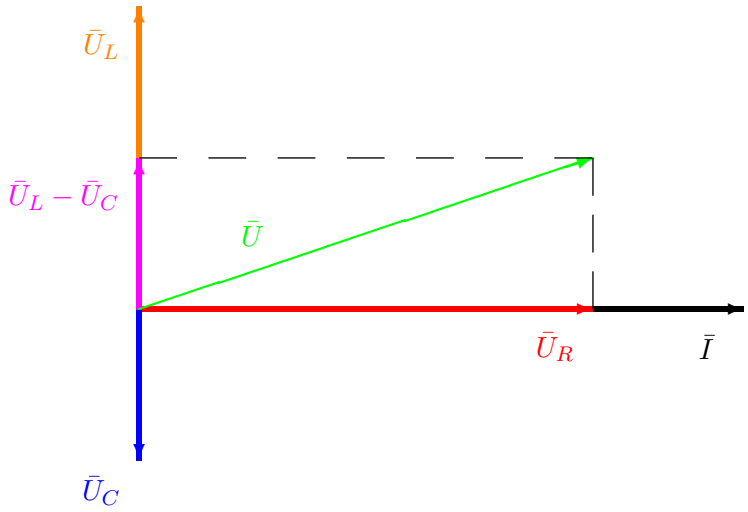
Obrázek 3.1: Časový průběh napětí
Zdroj: [10]



Obrázek 3.2: Schéma sériového RLC obvodu
Zdroj: Vlastní

Sériový RLC obvod

Schéma obvodu je na obrázku 3.2. Napětí na rezistoru U_R je ve fázi s proudem I. Napětí na kondenzátoru U_C je zpožděno za proudem o 90°. Napětí na cívce U_L předbíhá proud o 90°. Celkové napětí U je rovno vektorovému součtu napětí U_R, U_L a U_C. Z fázorového diagramu na obrázku 3.3 je vidět, že pokud dojde k rezonanci, tak napětí na rezistoru U_R bude rovno celkovému napětí U. Rezonance je stav obvodu, kdy reaktance cívky a kondenzátoru je shodná. Fázor $U_L - U_C$ bude tedy nulový.



Obrázek 3.3: Fázorový diagram sériového RLC obvodu
Zdroj: Vlastní

3.2 Symbolicko-komplexní metoda

I když je možné vystačit pro praktické výpočty s představou rotujících časových vektorů (fázorů), dává se při praktických výpočtech přednost tzv. *symbolickému počtu v komplexní rovině*. Oproti matematice se pro imaginární svislou složku vektoru nepoužívá písmenko *i*, protože by se pletlo s okamžitou hodnotou proudu, ale písmenko *j*. Každý bod v komplexní rovině má reálnou (vodorovnou) a imaginární (svislou) souřadnici. [6]

Pro řešení obvodů touto metodou je nutné znát vztahy uvedené v tabulce 3.1 a umět počítat s komplexními čísly. Nevýhodou symbolicko-komplexní metody je, že obvod může obsahovat pouze *lineární prvky* (rezistory, kondenzátory a cívky).

Tabulka 3.1: Základní vztahy pro symbolicko-komplexní metodu

Char. veličiny	Mat. vyjádření
Kapacitní reaktance:	$\mathbf{X_C} = -j\frac{1}{\omega C}$
Induktivní reaktance:	$\mathbf{X_L} = j\omega L$
Proud kondenzátorem:	$\mathbf{I_C} = \frac{\mathbf{U_C}}{\mathbf{X_C}}$
Proud cívkou:	$\mathbf{I_L} = \frac{\mathbf{U_L}}{\mathbf{X_L}}$

Zdroj: [1]

Sériový RLC obvod

Příklad demonstruje užití symbolicko-komplexní metody. Schéma obvodu je stejné jako u příkladu pro řešení obvodů pomocí fázorových diagramů - viz strana 9. Podle druhého Kirchhoffova zákona platí pro obvod vztah

$$\mathbf{U} = \mathbf{U_R} + \mathbf{U_L} + \mathbf{U_C}$$

kde U je vstupní napětí [V]
U_R je úbytek napětí na rezistoru R [V]
U_L je úbytek napětí na cívce L [V]
U_C je úbytek napětí na kondenzátoru C [V]

vyjádříme-li napětí na součástkách jako součin proudů a impedancí, můžeme psát

$$\mathbf{U} = R\mathbf{I} + jX_L\mathbf{I} - jX_C\mathbf{I}$$

kde U je vstupní napětí [V]
R je odpor rezistoru [Ω]
I je proud obvodem [A]
j je označení pro imaginární složku [-]
X_L je induktivní reaktance [Ω]
X_C je kapacitní reaktance [Ω]

z toho

$$\mathbf{U} = \mathbf{I}\left(R + j\omega L - j\frac{1}{\omega C}\right).$$

kde	U je vstupní napětí	[V]
	I je proud obvodem	[A]
	R je odpor rezistoru	[Ω]
	j je označení pro imaginární složku	[-]
	ω je úhlová frekvence	$[rad \cdot s^{-1}]$
	L je indukčnost	[H]
	C je elektrická kapacita	[F]

3.3 Diferenciální rovnice

Pokud máme nástroje pro řešení soustavy diferenciálních rovnic (např. TKSL, SIMLIB), tak je tato metoda jednoduchá. K sestavení rovnice je nutné znát základní vztahy, které jsou uvedeny v tabulce 3.2. Následně sestavujeme rovnice stejným způsobem jako u stejnosměrných obvodů.

Tabulka 3.2: Základní vztahy pro sestavení diferenciálních rovnic

Char. veličiny	Mat. vyjádření
Napětí na odporu:	$u_R = R \cdot i_R$
Proud odporem:	$i_R = \frac{1}{R} \cdot u_R$
Napětí na kondenzátoru:	$u_C = \frac{1}{C} \int i_C \, \mathrm{d}t$
Proud kondenzátorem:	$i_C = C \cdot u'_C$
Napětí na cívce:	$u_L = L \cdot i'_L$
Proud cívkou:	$i_L = \frac{1}{L} \int u_L \, \mathrm{d}t$

Zdroj: [6]

Sériový RLC obvod

Postupně provedeme odvození diferenciálních rovnic popisující obvod na obrázku 8.4. Nejprve vyjádříme proud i tekoucí obvodem. Protože víme podle druhého Kirchhoffova zákona, že napětí na cívce L je rovno $u_1 - u_C - u_2$, kde u_1 je napětí na vstupu, u_C je úbytek napětí na kondenzátoru C a u_2 je úbytek napětí na rezistoru R a zároveň i výstupní napětí. Můžeme psát podle vzorce pro výpočet proudu cívkou uvedeného v tabulce 3.2:

$$i' = \frac{1}{L}(u_1 - u_C - u_2).$$

Dále je nutné vyjádřit chybějící úbytek napětí na kondenzátoru C, opět využijeme vztahu pro výpočet napětí na kondenzátoru z tabulky 3.2:

$$u'_C = \frac{1}{C} i.$$

A nakonec vypočítáme napětí na rezistoru R jako

$$u_2 = iR.$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB lze najít v podkapitole 8.4.

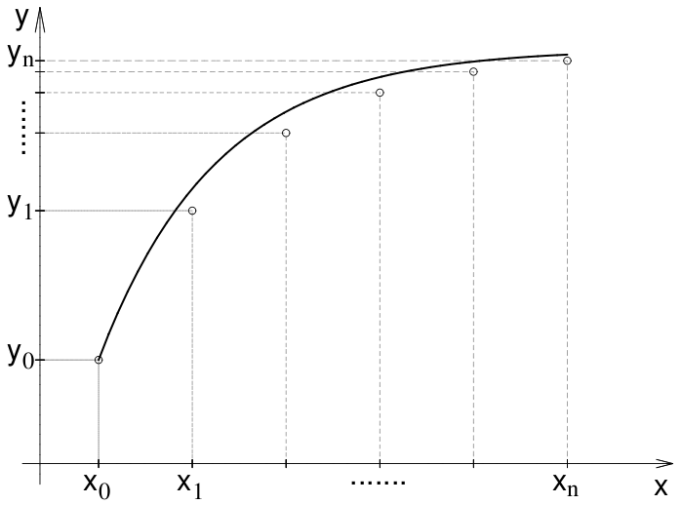
Kapitola 4

Metody pro numerické řešení lineárních diferenciálních rovnic

Diferenciální rovnice jsou jednou z možností, jak popsat a řešit elektronické obvody. Protože ve svém programu využívám k řešení obvodu právě diferenciální rovnice, tak je vhodné se jimi zabývat podrobněji. Diferenciální rovnice lze řešit analyticky nebo numericky. Řešení složitějších rovnic analyticky je obtížné a v některých případech i nemožné. Proto se k jejich řešení používají metody přibližné - numerické.

Společným znakem všech dále uvedených metod je, že řešení nehledáme jako spojitou funkci definovanou na celém zkoumaném intervalu $\langle a, b\rangle$, ale hodnoty přibližného řešení počítáme pouze v konečném počtu bodů $a = x_0 < x_1 < ... < x_n = b$. Těmto bodům říkáme *uzlové body*. Rozdíl $h_i = x_{i+1} - x_i$ se nazývá *krok*. Na velikost kroku záleží jak moc přesně se přiblížíme k řešení. Chceme-li znát přibližnou hodnotu řešení v jiném než uzlovém bodě, můžeme použít některou z interpolačních metod, např. nahradit řešení lomenou čarou procházející uzlovými body. [4]

Na obrázku 4.1 je vidět přesné řešení, které je vykresleno plnou čarou a přibližné hodnoty řešení v uzlových bodech - vyznačeno kroužky.



Obrázek 4.1: Přesné a přibližné řešení diferenciální rovnice
Zdroj: [4]

4.1 Jednokrokové metody

Jednokrokové metody jsou metody, které počítají přibližnou hodnotu řešení v dalším uzlovém bodě pomocí hodnoty v předchozím uzlovém bodě.

4.1.1 Eulerova metoda

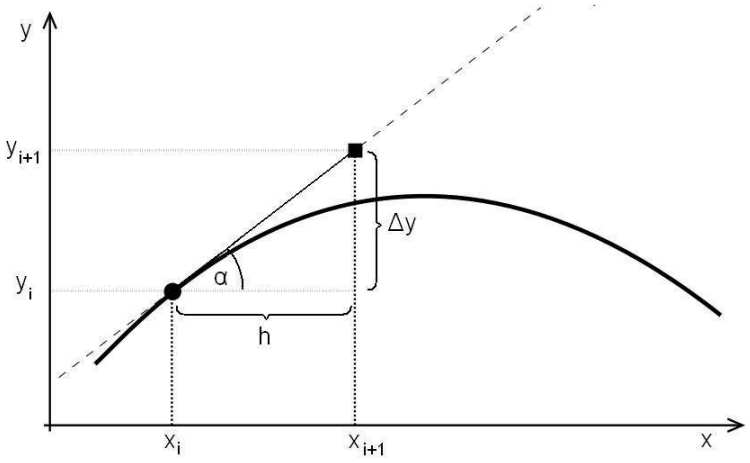
Nejjednodušší jednokroková metoda. Další uzlový bod se vypočítá ze vztahu

$$y_{i+1} = y_i + hy'_i, \tag{4.1}$$

kde y_i je přibližné řešení v předchozím uzlovém bodě,
h označuje integrační krok a
y'_i je derivace v předchozím uzlovém bodě.

Hodnotu řešení v bodě x_0 známe z počáteční podmínky. Rozumným zkracováním kroku lze dosáhnout vyšší přesnosti. Je nutné brát v úvahu, že v výpočet každého kroku vychází z hodnot kroku předešlého, který je již zatížen chybou z předchozích výpočtů. Proto nemusíme extrémně malým krokem dosáhnout vyšší přesnosti.

Odvození



Obrázek 4.2: Odvození Eulerovy metody
Zdroj: [7]

Z obrázku 4.2 je vidět, že platí rovnice:

$$tg\alpha = \frac{\Delta y}{h}$$

Po vyjádření Δy dostáváme

$$\Delta y = h \cdot tg\alpha$$

Geometrickou interpretací derivace je směrnice přímky, tedy:

$$\Delta y = h \cdot y_i'$$

K vyjádření y_{i+1} zbývá jen sečíst y_i a Δy:

$$y_{i+1} = y_i + h y_i'$$

Naše odvození se shoduje s rovnici 4.1 uvedenou v předchozím textu.

Příklad výpočtu

Na jednoduché diferenciální rovnici demonstruji použití Eulerovy metody. Mějme danou diferenciální rovnici:

$$y' = 2x, \quad y(0) = 0$$

Přibližné hodnoty řešení budeme počítat podle vzorce 4.1. Integrační krok jsem zvolil jako $h = 0,1$. Po úpravě dostáváme:

$$y_{i+1} = y_i + 0,1 \cdot 2x$$

Tabulka 4.1 ukazuje numerické řešení diferenciální rovnice v uzlových bodech pomocí Eulerovy metody s integračním krokem $h = 0,1$. Numerické řešení je značené jako y_i. Pro srovnání přesnosti metody je vhodné uvést i přesné analytické řešení, které označím jako $y(x_i)$. Analytickým řešením zadané diferenciální rovnice je rovnice $y = x^2$.

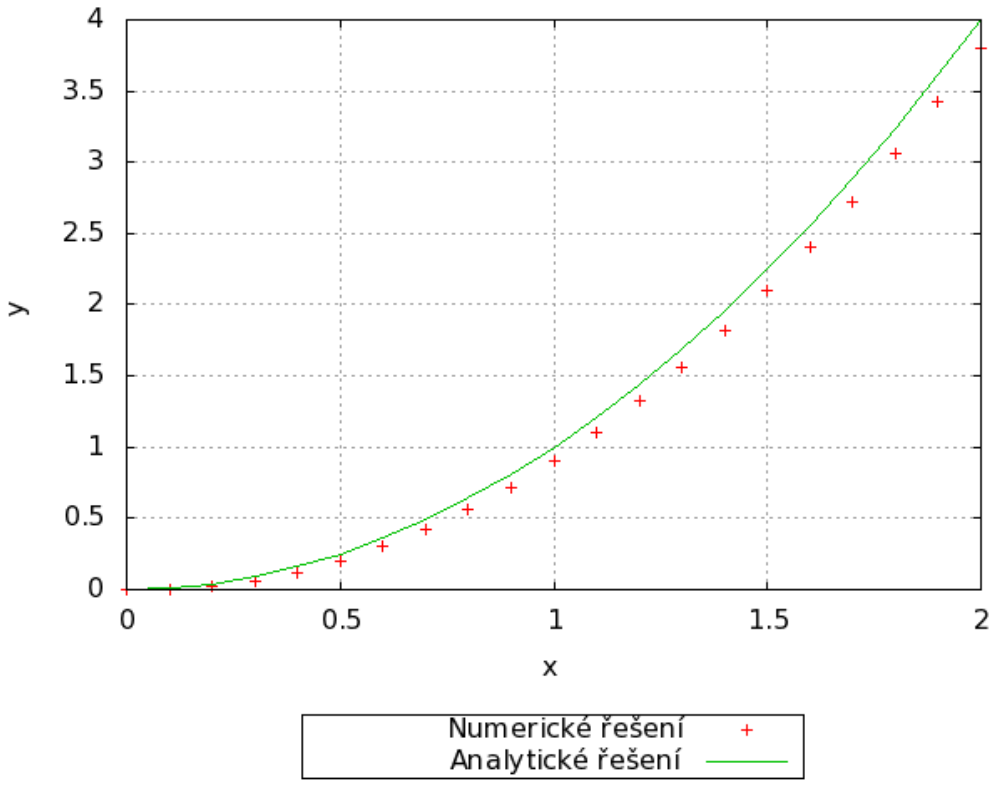
Tabulka 4.1: Srovnání numerického a analytického řešení

i	x_i	y_i	$y(x_i)$
0	0	0,00	0,00
1	0,1	0,00	0,01
2	0,2	0,02	0,04
3	0,3	0,06	0,09
4	0,4	0,12	0,16
5	0,5	0,20	0,25
6	0,6	0,30	0,36
7	0,7	0,42	0,49
8	0,8	0,56	0,64
9	0,9	0,72	0,81
10	1	0,90	1,00
11	1,1	1,10	1,21
12	1,2	1,32	1,44
13	1,3	1,56	1,69
14	1,4	1,82	1,96
15	1,5	2,10	2,25
16	1,6	2,40	2,56
17	1,7	2,72	2,89
18	1,8	3,06	3,24

i	x_i	y_i	$y(x_i)$
19	1,9	3,42	3,61
20	2	3,80	4,00

Zdroj: Vlastní

Pro lepší názornost jsem obě řešení vynesl do grafu 4.3. Vidíme, že numerické řešení není zcela přesné a během výpočtů dochází ke kumulaci chyby.



Obrázek 4.3: Srovnání numerického a analytického řešení
Zdroj: Vlastní

4.1.2 Metoda Runge-Kutta

Nejvíce používanou je metoda Runge-Kutta 4. řádu. Vztah pro výpočet následujícího uzlového bodu je:

$$
\begin{aligned}
y_{n+1} &= y_n + \frac{1}{6}h(k_1 + 2k_2 + 2k_3 + k_4) \\
k_1 &= f(x_n, y_n) \\
k_2 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_1) \\
k_3 &= f(x_n + \frac{1}{2}h, y_n + \frac{1}{2}hk_2) \\
k_4 &= f(x_n + h, y_n + hk_3)
\end{aligned} \tag{4.2}
$$

kde y_n je přibližné řešení v předchozím uzlovém bodě,
h označuje integrační krok,
f je derivace v předchozím uzlovém bodě a
k_1, k_2, k_3, k_4 jsou pomocné výpočty prováděné uvnitř kroku.

Metoda Runge-Kutta provádí uvnitř kroku další pomocné výpočty, čímž dosahuje vyšší přesnosti. Při stejné délce kroku je podstatně přesnější než Eulerova metoda.

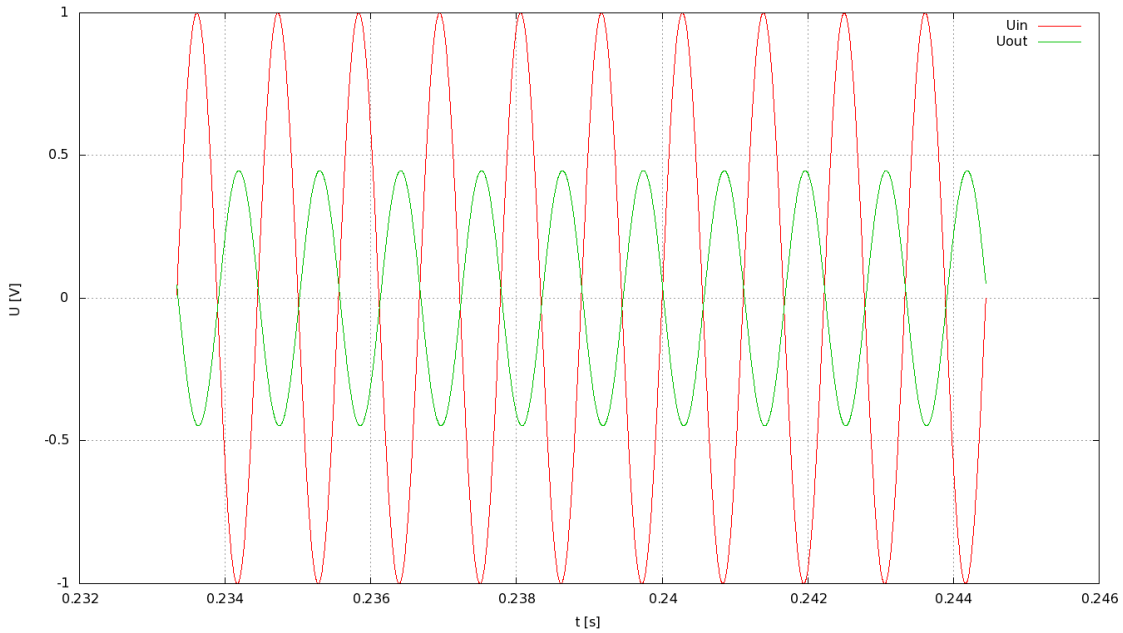
4.2 Vícekrokové metody

Vícekrokové metody počítají přibližnou hodnotu řešení v dalším uzlovém bodě pomocí hodnot v několika předchozích uzlových bodech. Proto je potřeba použít na začátku výpočtu jednokrokové metody, abychom si vypočítali několik „předchozích“ bodů. Tato vlastnost velmi komplikuje použití v simulacích při častých změnách integračního kroku. Vícekrokové metody jsou přesnější než jednokrokové metody, ale jejich výpočet je náročnější. Příkladem vícekrokové metody je metoda Adams-Bashforth.

Kapitola 5

Vyšetřování frekvenčních charakteristik a jejich konstrukce

Frekvenční charakteristiky zachycují závislosti obvodových veličin na frekvenci. Frekvenční charakteristiky nejčastěji vyjadřujeme graficky. Zvlášť amplitudovou a zvlášť fázovou charakteristiku. Frekvence se zobrazuje na vodorovnou osu a sledovaná veličina na svislou. Na frekvenční ose se často používá logaritmické měřítko.



Obrázek 5.1: Zobrazení průběhů napětí na osciloskopu
Zdroj: Vlastní

Nejprve si ukážeme postup při měření se skutečným obvodem. Až na konci kapitoly se budeme zabývat vyšetřováním charakteristik na počítači. Postup nalezení příslušných frekvenčních charakteristik elektrického obvodu je následující:

1. Zapojit obvod a připojit zdroj napětí.

2. Protože potřebujeme změřit i fázový posun mezi vstupním a výstupním napětím, nevystačíme si s voltmetry, ale potřebujeme dvoukanálový osciloskop. Jedním kanálem měříme průběh napětí na vstupu, druhým na výstupu. Dvoukanálový osciloskop umožňuje zobrazit průběhy obou napětí na jednu obrazovku a tím můžeme odečíst fázový posun. Situaci ilustruje obrázek 5.1, kde U_{in} je vstupní napětí a U_{out} výstupní napětí. V případě, že nechceme měřit fázi, tak stačí na změření napětí použít voltmetry.

3. Odečíst ustálené hodnoty vstupního napětí.

4. Odečíst ustálené hodnoty výstupního napětí.

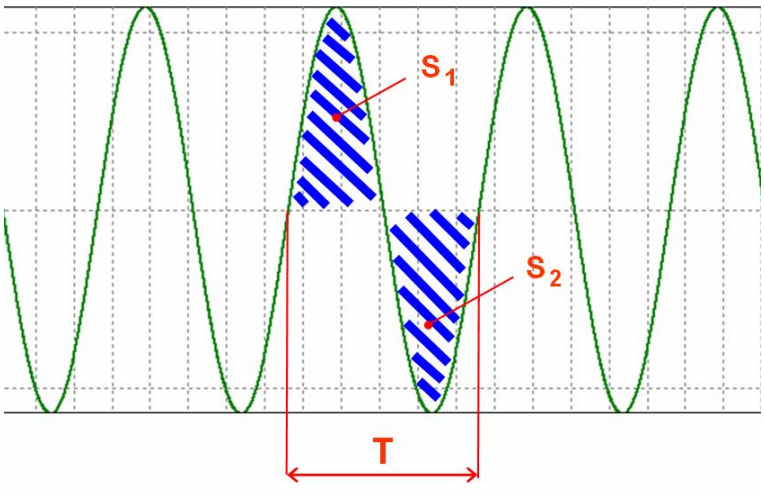
5. Výpočet amplitudového přenosu jako podíl výstupního napětí a vstupního napětí.

6. Odečtení časového posunu výstupního napětí vůči vstupnímu.

7. Výpočet fázového posunu.

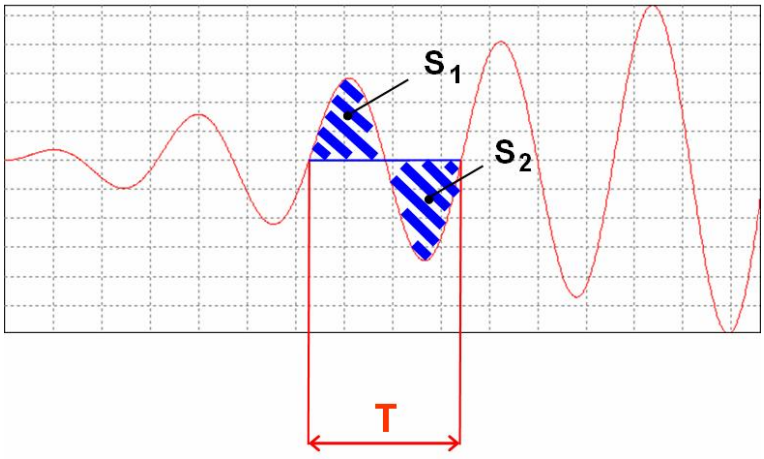
8. Vynesení získaných hodnot do grafů.

9. Posun frekvence vstupního signálu.



Obrázek 5.2: Ustálený harmonický signál
Zdroj: [10]

Přesnost frekvenční charakteristiky ovlivňuje zejména přesnost odečtení hodnoty amplitudy výstupního napětí a fázového posunu. Ale je nutné provádět odečtení hodnot v takovém místě, kde je výstupní napětí ustáleno. Rychlost ustálení je závislá na frekvenci vstupního napětí a parametrech obvodu. Tento problém ilustruje obrázek 5.2, kde můžeme vidět ustálený harmonický signál a obrázek 5.3, kde je zobrazen signál neustálený. Pro ustálený harmonický signál platí, že kladná i záporná půlvlna musí mít stejnou plochu, tedy $|S_1| = |S_2|$. Naopak pokud signál není ustálen, tak jsou plochy $|S_1|$ a $|S_2|$ rozdílné. Obecně můžeme konstatovat, že určitý integrál přes jednu periodu ustáleného signálu má hodnotu 0. Není-li signál ustálen je tato hodnota nenulová.



Obrázek 5.3: Neustálený signál
Zdroj: [10]

5.1 Amplitudová charakteristika

Vyjadřuje závislost přenosu na frekvenci. Přenos se vypočítá jako poměr výstupní a vstupního napětí. Pokud není přenos přepočítán na decibely, tak se jedná se o bezrozměrnou veličinu, tedy

$$A_U = \frac{U_2}{U_1} \tag{5.1}$$

kde A_U je přenos filtru [-]
U_1 je vstupní napětí [V]
U_2 je výstupní napětí [V]

Někdy se přenos vyjadřuje v decibelech podle vztahu

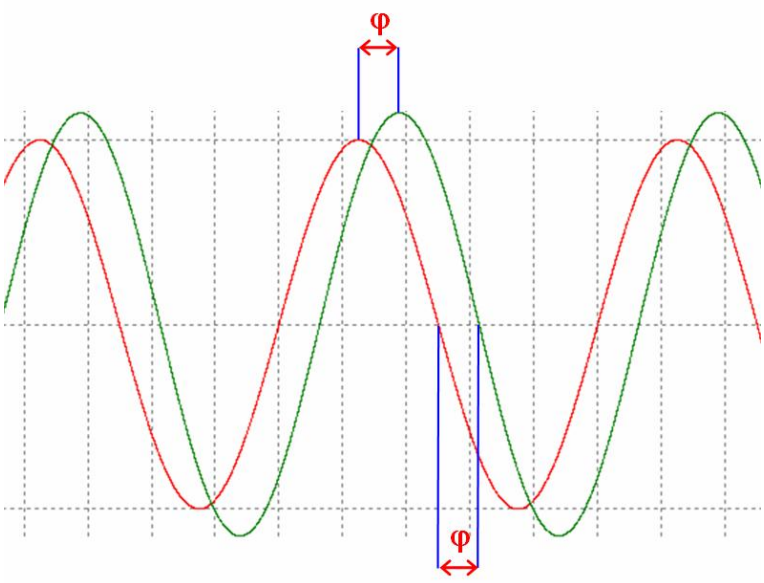
$$A_{UdB} = 20 \cdot log\left(\frac{U_2}{U_1}\right) \tag{5.2}$$

kde A_{UdB} je přenos filtru v decibelech [dB]
U_1 je vstupní napětí [V]
U_2 je výstupní napětí [V]

Příklady amplitudových charakteristik jsou na obrázcích 2.1 a 6.4.

5.2 Fázová charakteristika

Zachycuje závislost fázového posunu na frekvenci. K odečtení fáze lze použít již dříve nalezená maxima (myšleno jejich časy) nebo je možné zjistit časy průchodů signálů nulou. Způsoby odečtení fáze jsou na obrázku 5.4. Oba způsoby jsou ekvivalentní.



Obrázek 5.4: Možnosti odečtení fázového posunu
Zdroj: [10]

Máme-li odečteny z osciloskopu všechny potřebné hodnoty, můžeme fázi vypočítat pomocí vztahu:

$$\varphi = \frac{-360}{T}\Delta t = -360 f \Delta t$$

kde	T je perioda vstupního signálu	[s]
	f je frekvence vstupního signálu	[Hz]
	Δt je časový posun výstupního napětí	[s]

Při měření časového posunu je třeba dbát na jeho znaménko. Pokud se výstupní signál zpožďuje, je znaménko kladné, pokud předbíhá je znaménko záporné. Protože Δt násobíme zápornou konstantou -360, tak pro fázi je to přesně naopak. Předbíhá-li výstupní napětí vstupní, tak je fáze kladná, pokud je výstupní napětí zpožděno, tak je fáze záporná. Obecně tedy platí, že $\Delta t = (t_2 - t_1)$, kde t_2 je čas odečtení hodnoty maxima výstupního napětí nebo jeho průchodu nulou a t_1 je čas odečtení vstupního napětí.

Příklady fázových charakteristik jsou na obrázcích 2.2 a 6.5.

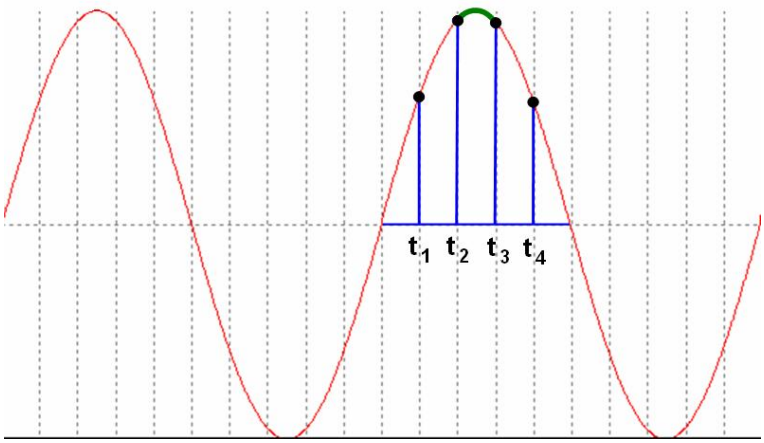
5.3 Vyšetřování charakteristik na počítači

Simulace na počítači s sebou přináší několik problémů. Problémy vyplývají ze samotné podstaty. Kdy se snažíme řešit na počítači, který pracuje diskrétně, spojitý systém (elektrický obvod).

Jedná se o problémy lokalizace význačných bodů:

- maxima (viz obrázek 5.5) nebo
- průchodu signálu nulou

a s tím úzce související vhodnou volbu kroku při měření. Pokud zvolíme krok příliš velký, tak se může stát, že maximum „přeskočíme“. Zvolíme-li krok příliš malý, bude simulace trvat příliš dlouhou dobu. Krok je definován tzv. *frekvenci vzorkování*. Frekvenci vzorkování nastavuji při tvorbě simulačního modelu (viz kapitola 6.2.3). V mé implementaci simulátoru probíhá vzorkování 500-krát v rámci každé periody signálu.



Obrázek 5.5: Problém lokalizace maxima
Zdroj: [10]

Postup vyšetřování charakteristik na počítači je analogický postupu při měření se skutečným obvodem:

- Sestavit simulační model. Metody řešení elektronických obvodů byly diskutovány v kapitole 3.
- Spustit simulaci pro konkrétní frekvenci.
- Odečíst ustálenou hodnotu vstupního napětí.
- Odečíst ustálené hodnoty výstupního napětí.
- Výpočet amplitudového přenosu jako podíl výstupního napětí a vstupního napětí.
- Odečtení časového posunu výstupního napětí vůči vstupnímu.
- Výpočet fázového posunu.
- Vynesení získaných hodnot do grafů.
- Posun frekvence vstupního signálu.

Detailní ukázka algoritmu je v podkapitole 6.4.6.

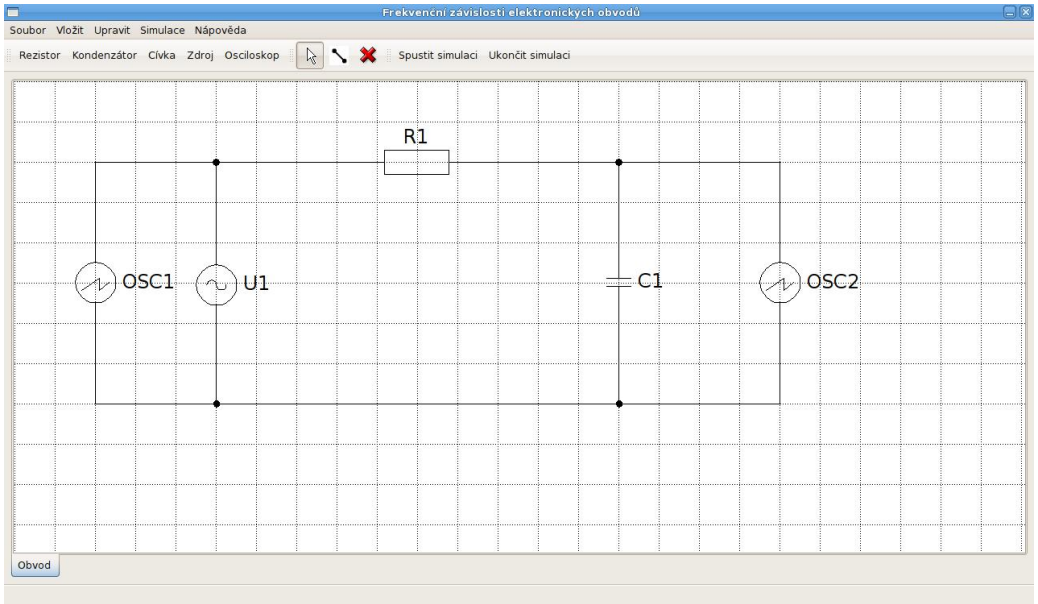
Kapitola 6

Popis řešení simulátoru střídavých elektronických obvodů

Praktická část této práce se zabývá implementací programu pro automatizované vyšetřování frekvenčních charakteristik. Program je napsán v jazyce C++ s využitím pomocných knihoven popsaných v podkapitole 6.2.

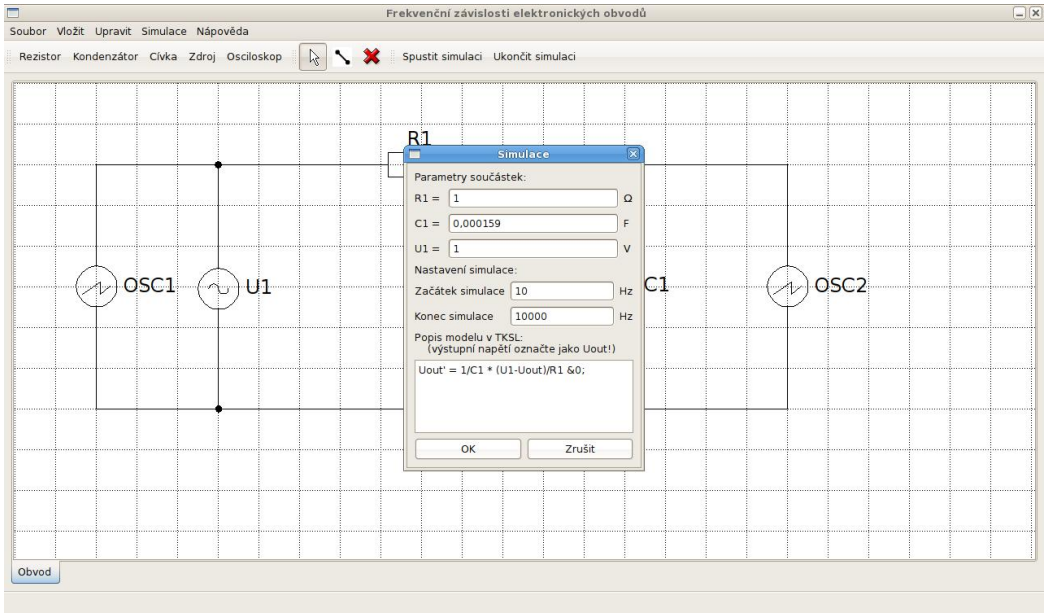
6.1 Ovládání programu

Obrázek 6.1: Po spuštění programu uživatel vidí editor, kam zadá schéma zkoumaného elektronického obvodu. Součástky a spoje uživatel přidává z horního menu. Obrázek ilustruje situaci po zapojení celého elektronického obvodu. Připojené osciloskopy plní pouze pedagogickou funkci, aby si student uvědomil, jak probíhá měření se skutečným obvodem. K činnosti programu nejsou potřeba.



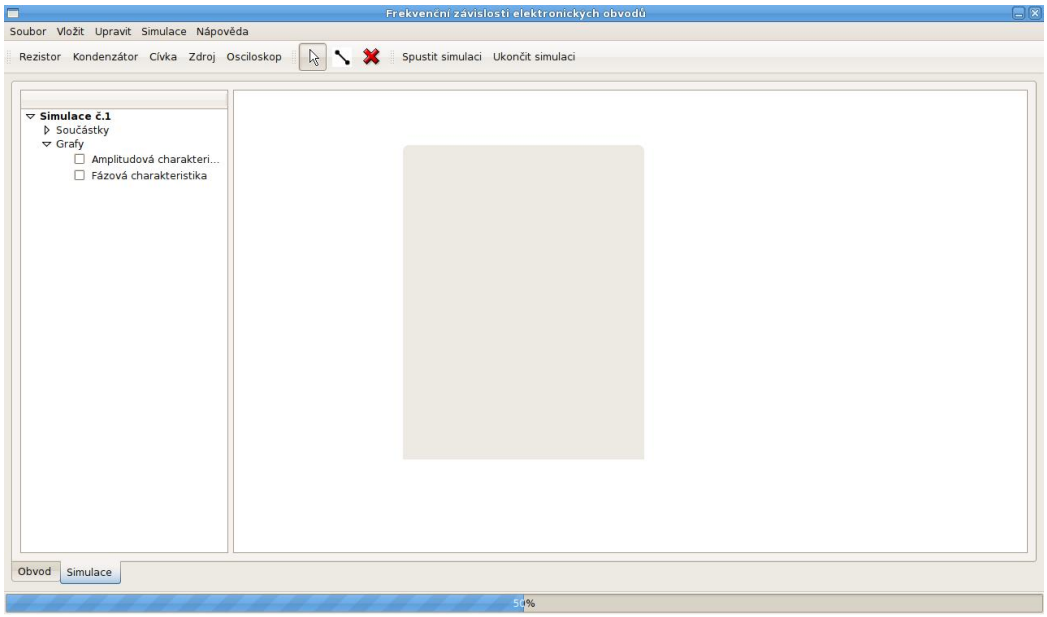
Obrázek 6.1: Hlavní okno programu s editorem pro schématické zadávání el. obvodů
Zdroj: Vlastní

Obrázek 6.2: Po kliknutí na tlačítko „Spustit simulaci“ z horního menu, se uživateli zobrazí dialogové okno pro nastavení parametrů simulace.



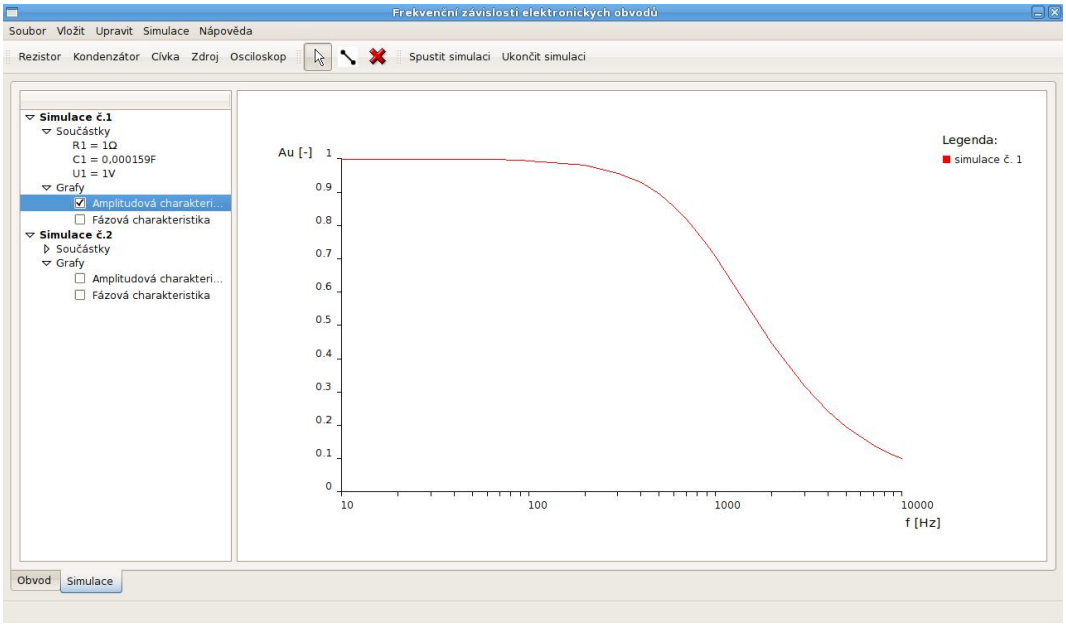
Obrázek 6.2: Dialogové okno pro zadání parametrů simulace
Zdroj: Vlastní

Obrázek 6.3: Průběh výpočtu je uživateli indikován pomocí progress baru ve spodní části okna programu.



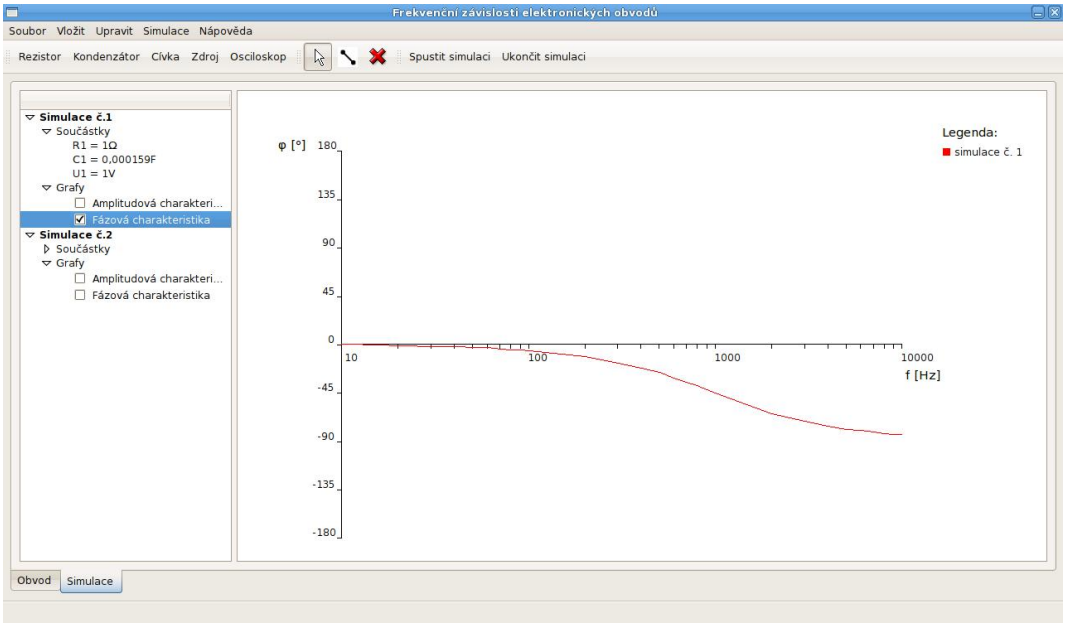
Obrázek 6.3: Průběh vyšetřování charakteristik
Zdroj: Vlastní

Obrázek 6.4: Amplitudové charakteristiky lze vykreslit zatrhnutím volby ve výpisu vlevo.



Obrázek 6.4: Zobrazení amplitudové frekvenční charakteristiky
Zdroj: Vlastní

Obrázek 6.5: Stejným způsobem je možno zobrazit i fázové charakteristiky.



Obrázek 6.5: Zobrazení fázové frekvenční charakteristiky
Zdroj: Vlastní

Simulace je možné spouštět opakovaně a jejich výsledky zobrazit do jednoho grafu. Parametry součástek, se kterými byla simulace spuštěna, lze vypsat po rozkliknutí podmenu „Součástky“.

6.2 Pomocné knihovny

Program je grafickou aplikací napsanou v jazyce C++ za použití frameworku Qt 4.7.0. Program, po spuštění simulace uživatelem, vytvoří simulační model opět v jazyce C++ s použitím knihovny SIMLIB. Systém TKSL je v seznamu knihoven uveden pro úplnost. Systém TKSL pro běh programu není potřeba. Využívám jen přehlednější syntaxe TKSL oproti SIMLIB pro popis obvodu diferenciálními rovnicemi.

6.2.1 Qt

Qt (vyslovuje se jako anglické cute (roztomilý), tedy [kju:t]) je jednou z nejpopulárnějších multiplatformních knihoven pro vytváření programů s grafickým uživatelským rozhraním.

Autory frameworku jsou norští programátoři Haavard Nord a Eirik Chambe-Eng. Na první verzi Qt začali pracovat již v roce 1991 a od té doby se framework stále rozšiřuje. Následujícího roku přišel Eirik s myšlenkou „signálů a slotů“. Jednoduchým ale mocným GUI programovacím paradigmatem, které nyní využívají i jiné toolkity. O 3 roky později společně oba vývojáři založili společnost Trolltech. Zajímavostí je vznik názvu Qt. Písmeno `Q` bylo vybráno jako prefix všech tříd jenom proto, že hezky vypadalo v editoru Emacs. Písmeno `t` znamená „toolkit“. V roce 2008 firma Trolltech prodala Qt společnosti Nokia. Celý framework je dostupný zdarma pod licencí GPL nebo za poplatek pro komerční použití. Qt je knihovna programovacího jazyka C++, ale existuje i pro jiné jazyky jako Python (PyQt), Ruby (QtRuby) a jiné. [2]

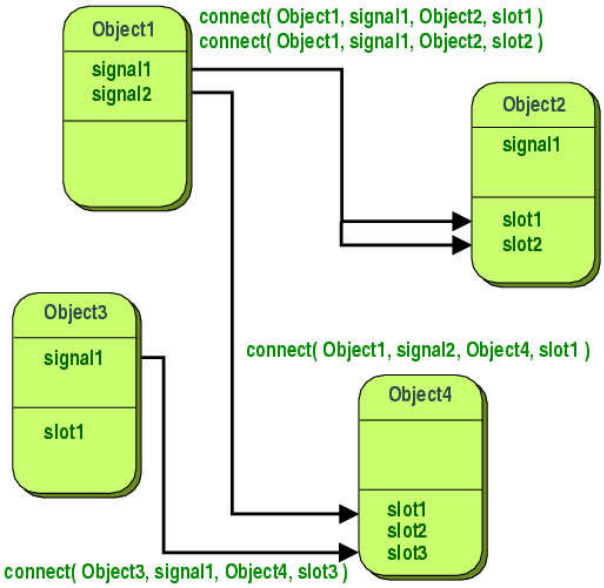
Velkou výhodou je Qt je velmi přehledně zpracovaná dokumentace, která je dostupná online na [9], a také vývojové prostředí Qt Creator. Aplikace vytvořené v Qt pro grafické uživatelské prostředí používají nativní vzhled OS, takže se aplikace vždy přizpůsobí do používaného prostředí.

Qt používá například webový prohlížeč Opera, populární linuxové desktopové prostředí KDE, komunikátor Skype, virtualizační software VirtualBox a další aplikace. Qt běží i na nejrůznějších mobilních zařízeních. Podpora nejen pro unixové operační systémy jako Linux a BSD, ale i Windows a Mac OS X je samozřejmostí.

Signály a sloty

Signály a sloty slouží ke komunikaci mezi objekty Qt. Například při kliknutí uživatele na tlačítko „Zavřít“ chceme, aby bylo zavolána funkce pro zavření okna. Mechanismus signálů a slotů je jednou z nejdůležitějších vlastností Qt.

Sloty a signály mohou být využity ve všech objektech, které jsou přímo nebo nepřímo zděděny ze třídy QObject. Signál je vyslán, pokud dojde k jisté události. Signál je zvláštní metoda, která nemá tělo a nevrací hodnotu, má pouze prototyp. Widgety v Qt mají množství předdefinovaných signálů, ale není problém vytvářet i signály vlastní. Slot je metoda, která je zavolána v reakci na vyslaný signál. Jedná se v podstatě o obyčejnou metodu, kterou můžeme zavolat i sami. Jak vyplývá z obrázku 6.6, jeden slot může reagovat na více signálů a zároveň jeden signál může být napojen na několik slotů.



Obrázek 6.6: Signály a sloty v Qt
Zdroj: [9]

K propojení objektů slouží metoda `bool QObject::connect ( const QObject * sender, const char * signal, const QObject * receiver, const char * method, Qt::ConnectionType type = Qt::AutoConnection )`.

Parametry metody jsou:

1. Ukazatel na instanci objektu Qt, který vyslal signál.
2. Signál, na který bude daný slot reagovat.
3. Ukazatel na instanci objektu Qt, jehož slot se spustí v reakci na vyslaný signál. Argument lze vynechat, pak se bude metoda chovat jako by za tento parametr dosadilo klíčové slovo `this`.
4. Slot, který se spustí.
5. Typ propojení. Určuje zda je signál doručen slotu okamžitě nebo až někdy později.

6.2.2 TKSL

TKSL (Taylor-Kunovský Simulation Language) je určen k simulaci dynamických systémů, systémů s nespojitostmi a systémů popsaných parciálními diferenciálními rovnicemi. Veškeré výpočty jsou založeny na diferenciálních rovnicích. Systém provádí numerické řešení pomocí Taylorova rozvoje. Požadovaná přesnost je udržována dynamickým nastavováním řádu Taylorova polynomu (až 100 i více) a v současné době přesnost řádově 10-20 není vůbec výjimečná. Vstupem je tedy soustava diferenciálních rovnic (maximálně 150) popisující simulovanou soustavu. Program je určen pro operační systém MSDOS a je nenáročný na hardware. V současné době je dokončována nová verze simulačního jazyka naprogramovaná v jazyce C - TKSL/C. [5]

Struktura programu

Zdrojový kód programu v TKSL může vypadat následujícím způsobem:

```
var
  y,z,A,B,C;                    { deklarace proměnných }
const
  b0=2, b1=1, b2=1, b3=2,          { definice konstant }
  a0=1, a1=1, a2=2,
  tmax = 10, DT=0.1, EPS = 1e-20;
system
  A'= b0*z - a0*y     &0; { zápis jednotlivých rovnic  }
  B'= b1*z - a1*y + A &0; { za & je počáteční podmínka }
  C'= b2*z - a2*y + B &0;
  y = b3*z + C;
  z = 1;
sysend;
```

[5]

Program je sestaven z několika sekcí:

1. Deklarace proměnných = uvozena klíčovým slovem `var` a ukončena středníkem. Proměnné jsou odděleny čárkami.
2. Definice konstant = uvozena klíčovým slovem `const` a ukončena středníkem. Jednotlivé konstanty jsou odděleny čárkami. Jména konstant tmax, DT a EPS jsou rezervována pro stanovení simulačního času, integračního kroku a přesnosti výpočtu.
3. Tělo programu = píše se mezi klíčová slova `system` a `sysend`. Zápis rovnic odpovídá jejich matematickému zápisu. Jen s tím rozdílem, že počáteční podmínky se značí `&` a rovnice jsou odděleny středníkem.
4. Ukončení programu = uvozeno klíčovým slovem `sysend`.
5. Komentáře = komentáře se píší mezi složené závorky.

Nejvíce nás bude zajímat blok těla programu, protože stejný zápis je zadán na vstup mého programu. Ačkoliv program ke své činnosti využívá simulační knihovny SIMLIB, tak rovnice jsou zadávány právě v TKSL. Zejména kvůli srozumitelnosti a jednoduchosti jejich zápisu.

6.2.3 SIMLIB

Simulační knihovna SIMLIB je vyvíjena od roku 1990 na Ústavu informatiky a výpočetní techniky FEI VUT Brno. Knihovna je implementována na počítačích třídy PC pod MS-DOS (překladače Borland C++ nebo GNU C++) a pod operačním systémem Linux (překladač GNU C++). Knihovna je přenositelná i na jiné platformy, vyžaduje však úpravu v modulu implementujícím procesy. SIMLIB poskytuje základní prostředky pro popis spojitých, diskrétních i kombinovaných modelů a prostředky pro řízení simulace. Při tvorbě simulačních modelů a experimentování s nimi lze použít různá integrovaná prostředí, která umožňují interaktivní tvorbu a ladění modelů. Knihovna usnadňuje efektivní popis modelů přímo

v jazyce C++, není tedy nutný překladač simulačního jazyka. Tato koncepce má své výhody i nevýhody. Je možné používat všech ostatních prostředků vytvořených v C++ (např. grafické knihovny a uživatelská rozhraní). Uživatel také není nijak omezován při případném doplňování prostředků knihovny. Za nevýhodu lze považovat nemožnost dodatečných syntaktických a sémantických kontrol, které by se mohly provádět při použití překladače simulačního jazyka. Na straně uživatele se předpokládá základní znalost programování v jazyce C++. [8]

Typy bloků

Základní výčet bloků důležitých pro spojitou simulaci. Úplný seznam bloků lze najít v dokumentaci [8].

- Constant - konstanty, nelze je měnit.
- Variable - proměnné, lze je měnit kdykoli.
- Expression - blokový výraz.
- Parameter - parametry, lze je měnit jen mezi simulačními běhy.
- Integrator - slouží k implementaci integračního mechanismu spojité simulace. Nastavení počáteční podmínky je druhým parametrem konstruktoru objektu, tedy například:

```
...
Integrator UC;
...
UC(I/C, pocatecni_podminka),
...
```

Řízení simulace

- Sampler - zajišťuje periodické volaní funkce. Typicky funkce odpovědné za výpočet nových dat ze simulace.
- Print(...) - tisk hodnot na výstup.
- SetOutput(filename) - přesměrování výstupu do zadaného souboru.
- SetStep(krok) - funkce nastaví krok.
- Init(t0, t1) - inicializace simulace. `t0` je počáteční čas a `t1` je koncový čas.
- Run() - spuštění simulace.

Příklad kompletního programu

Jedná se o příklad pro horní propust. Na vstup filtru je přivedeno sinusové napětí.

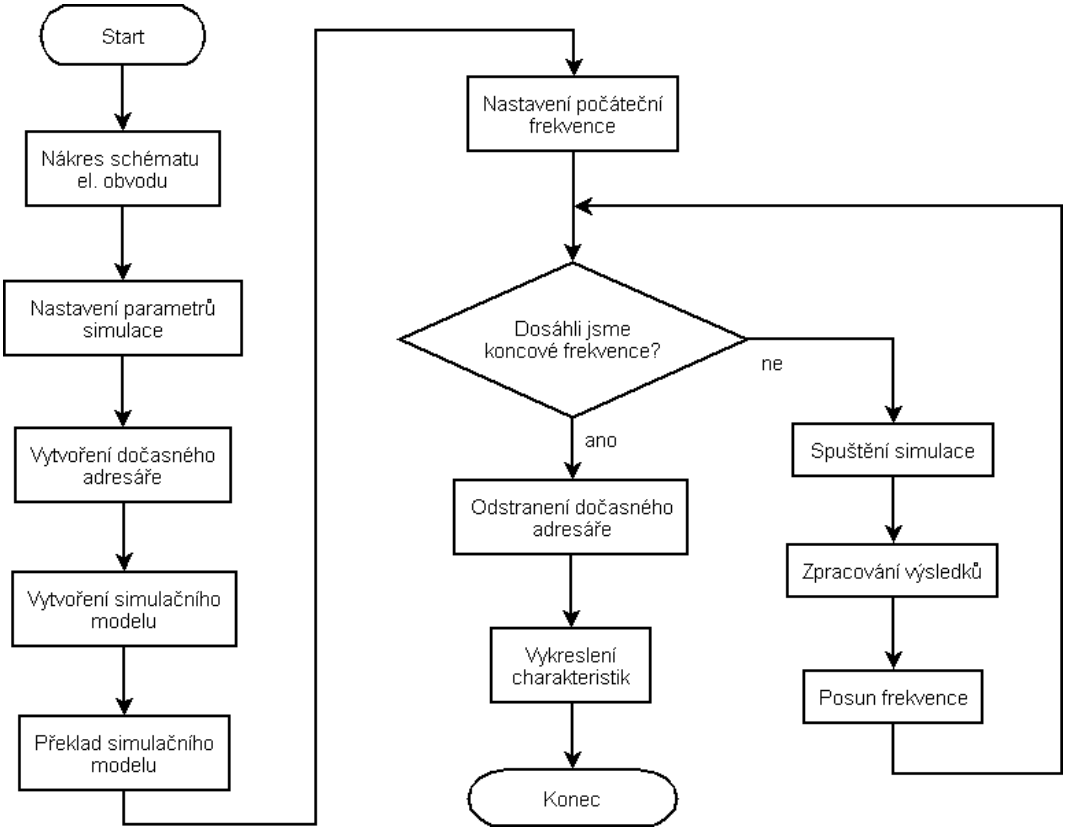
```
#include ,,simlib.h‘‘
#include <stdio.h>
#include <stdlib.h>

#define PI 3.14159265358979323846264338327
double FREKVENCE = 0;

struct DERIVACNI_CLANEK{
    Variable U1;
    Parameter R, C;
    Expression I, U2;
    Integrator UC;

    DERIVACNI_CLANEK( double R, double C ):

        R(R), C(C),
        I(U2/R),
        UC(I/C, 0),
        U2(U1 - UC) {}
};

DERIVACNI_CLANEK filtr( 1, 0.000159 );

void Sample1() {
    Expression f(FREKVENCE);
    filtr.u1 = ((1*Sin(2*PI*f*T))).Value();
    Print(,,%g %g %g\n‘‘, T.Value(), filtr.u1.Value(), filtr.uout.Value());
}

Sampler S(Sample1, 0);

int main(int argc, char *argv[]){
    FREKVENCE = atof(argv[1]);
    S.SetStep((1/FREKVENCE)/500);       // frekvence vzorkovani
    Init(0, 1/FREKVENCE * 20);          // promerim 20 period
    Run();
    return 0;
}
```

Další příklady jsou uvedeny ve sbírce příkladů v kapitole 8.

6.3 Vývojový diagram činnosti programu

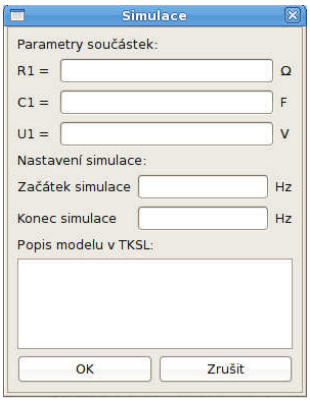
V podkapitole podrobně okomentuji vývojový diagram programu na obrázku 6.7.

- Nákres schématu elektrického obvodu = uživatel nakreslí schéma filtru.
- Nastavení parametrů simulace = po kliknutí na tlačítko „Spustit simulaci“, se uživateli zobrazí dialogové okno uvedené na obrázku 6.8. Zde vyplní



Obrázek 6.7: Vývojový diagram běhu programu
Zdroj: Vlastní

– parametry jednotlivých součástek,
– počáteční frekvenci a koncovou frekvenci,
– popis modelu v TKSL.



Obrázek 6.8: Dialogové okno pro nastavení simulace
Zdroj: Vlastní

- Vytvoření dočasného adresáře = v adresáři s programem se vytvoří dočasný adresář „simulace“, kam se uloží v dalším kroku vytvořený simulační model a kde se provede i jeho překlad.
- Vytvoření simulačního modelu = provede se převod z TKSL do SIMLIB. Převod zajišťuje konečný automat. Model je parametrizován. Má dva parametry. Jeho prvním parametrem je frekvence. V opačném případě by se musel pro každý běh simulace s jinou frekvencí opakovaně vytvářet a překládat znovu model, což by velice zpomalovalo simulaci. Druhým parametrem je počet period signálu, který vlastně přímo ovlivňuje dobu běhu jedné simulace. V rámci jedné simulace se musí opakovaně spouštět simulační model, aby se proměřila celá charakteristika.
- Překlad simulačního modelu = standardní překlad a přilinkování knihovny SIMLIB: `g++ -std=c++98 -o simulace simulace.cpp -lsimlib -lm`
- Nastavení počáteční frekvence = nastaví se počáteční frekvenci, kterou zadal uživatel při spuštění simulace.
- Spuštění simulace = spuštění simulačního modelu. Pokud výstupní signál není ustálen, spustí se simulace znovu s jinou délkou běhu (ve vývojovém diagramu 6.7 pro větší přehlednost digramu není zachyceno).
- Zpracování výsledků = výsledky se vypisují na standardní výstup procesu, který spustil simulaci. Výpis zajišťuje příkaz

  ```
  Print(,,%g %g %g\n‘‘, T.Value(), filtr.u1.Value(), filtr.uout.Value());}.
  ```

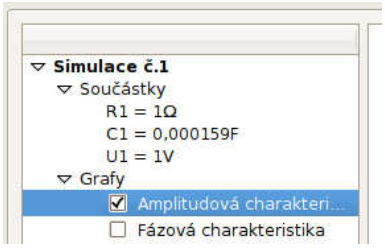
  Celý algoritmus pro zpracování výsledků a výpočtu přenosu a fáze je uveden v podkapitole 6.4.6.
- Posun frekvence = frekvenci se posouvá tak, abych se proměřily všechny klíčové body na logaritmické stupnici. Při každém nastavení nové frekvence je nutné hlídat, abychom se nedostali za koncovou frekvenci zadanou uživatelem.
- Odstranění dočasného adresáře = po dokončení běhu simulace se vymaže dočasně vytvořený adresář.
- Vykreslení charakteristik = uživatel si může vybrat mezi zobrazením amplitudové a fázové charakteristiky - viz obrázek 6.9. Výsledky několika simulací je možné zobrazit do jednoho grafu. Můžeme komfortně porovnávat chování filtru při změnách hodnot součástek atd.

6.4 Implementace vytvořeného programu

Kapitola bude věnována popisu všech tříd a jejich nejdůležitějších metod.

6.4.1 Třída MainWindow

Základní třída, která se jako jediná vytvoří v `main()` programu. Třída dědí z `QMainWindow`. Stará se o vykreslení hlavního okna programu. V konstruktoru třídy jsou volány metody:



Obrázek 6.9: Výpis parametrů simulace a možnost výběru mezi charakteristikami
Zdroj: Vlastní

- `MainWindow::vytvorAkce()` = metoda vytvoří akce a provede jejich napojení na odpovídající sloty.
- `MainWindow::vytvorMenu()` = metoda provede „navázaní“ akcí na příslušné položky v menu hlavního okna.
- `MainWindow::vytvorToolbar()` = činnost obdobná předchozí metodě. Jen s tím rozdílem, že vytvoří položky v toolbaru.
- `MainWindow::vytvorPracovniPlochu()` = metoda vytvoří nový objekt třídy `Editor`, pohled na editor a `QHBoxLayout`, který se stará o rozložení prvků na pracovní ploše. Pohled (třída `QGraphicsView`) musí mít zapnut tzv. tracking myši a nastaveno přeposílání zpráv na `Editor`.

Dále třída obsahuje implementaci slotů, na které jsou napojeny dříve vytvořené akce.

6.4.2 Třída Editor

Třída dědí z `QGraphicsScene`. V konstruktoru se vykreslí mřížka sloužící k pohodlnější práci s editorem a nastaví se mód na „nečinný“. `Editor` je implementován jako konečný automat, který má šest stavů: nečinný, vložitSoučástku, vložitSpoj, vložitSpojPokr, ukončitSpoj, pohybSoučástka a pohybSpoj. Jádrem třídy jsou následující funkce:

- `Editor::mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent)` = metoda je volána po stisku tlačítka na myši. Stiskne-li uživatel pravé tlačítko myši, tak je zobrazeno vyskakovací menu, které umožňuje otočit součástku po směru hodinových ručiček, otočit součástku proti směru hodinových ručiček nebo ji vymazat. Po stisku levého tlačítka se dostane program do konečného automatu. Metoda má implementováno chování pro následující stavy:
  - `necinny` = kontrola, zda-li uživatel neklikl na součástku nebo spoj. V případě že ano, tak dojde k přechodu do odpovídajícího stavu.
  - `vlozitSoucastku` = po kliknutí je součástka vložena na aktuální pozici a mód je nastaven na `necinny`.
  - `vlozitSpoj` = spoj může být vložen pouze po kliknutí na součástku nebo na jiný spoj. Při kliknutí na spoj jsou souřadnice vkládaného spoje přepočítány tak, aby vkládány spoj na jiný spoj plynule navazoval. Kolem spoje je implementován

jakýsi „koridor“. Uživatel tedy nemusí přesně kliknout na spoj, aby došlo k přichycení nového spoje. Při kliknutí na součástku jsou počáteční souřadnice vkládaného spoje přepočítány podle orientace součástky následovně:

1. součástka je orientovaná vodorovně, uživatel klikl do levé části součástky → souřadnice jsou přepočítány na levý okraj doprostřed.
2. součástka je orientovaná vodorovně, uživatel klikl do pravé části součástky → souřadnice jsou přepočítány na pravý okraj doprostřed.
3. součástka je orientovaná svisle, uživatel klikl do horní části součástky → souřadnice jsou přepočítány na horní okraj doprostřed.
4. součástka je orientovaná svisle, uživatel klikl do dolní části součástky → souřadnice jsou přepočítány na dolní okraj doprostřed.

– `vlozitSpojPokr` = opět pouze kontrola, jestli uživatel neklikl na součástku nebo spoj. Jinak není možné přidávání spoje ukončit.

– `ukoncitSpoj` = nastavení pomocných proměnných a přechod do stavu `necinny`.

- `Editor::mouseMoveEvent(QGraphicsSceneMouseEvent *mouseEvent)` = metoda je volána při pohybu myší. Má implementované chování pro čtyři stavy konečného automatu:

  – `vlozitSoucastku` = při vkládání součástky není potřeba držet tlačítko myši, součástka se pohybuje společně s kurzorem myši. Kolem každé čáry tvořící mřížku editoru je opět naimplementován „koridor“. Což znamená, že pokud se uživatel dostane se součástkou blízko k mřížce, tak je součástka k mřížce „přitáhnuta“. Toto řešení zvyšuje uživatelský komfort práce s editorem.

  – `vlozitSpojPokr` = k pokračování spoje je přidána další čára. Koncová souřadnice odpovídá pozici kurzoru myši. Na základě předchozí části spoje (každý spoj je tvořen několika čárami) je nastaven přidávané čáře úhel tak, aby každá čára byla rovná úsečka.

  – `pohybSoucastka` = součástka je v pohybu, takže se musí upravovat její pozice v závislosti na pohybu kurzoru. Opět jsou souřadnice přepočítávány tak, aby se součástka přichytila k mřížce. Dále je potřeba upravit pozice spojů, které jsou na součástku napojeny.

  – `pohybSpoj` = je zavolána metoda `Spoj::mouseMoveEvent(QPointF souradnice)`, která zajistí korektní pohyb se spojem.

- `Editor::mouseReleaseEvent(QGraphicsSceneMouseEvent *mouseEvent)` = metoda je volána po uvolnění tlačítka na myši. Má implementované chování pouze pro tři stavy konečného automatu. Pokud je automat ve stavu `ukoncitSpoj` a `pohybSoucastka`, tak dojde k přechodu do stavu `necinny` a je zavolána původní metoda ze třídy `QGraphicsScene`. Pokud je aktuální mód nastaven na `pohybSpoj` tak se musí zkontrolovat, zda-li uživatel po dokončení pohybu se spojem neklikl na součástku nebo spoj. V případě že ano, je spoj přichycen na onu součástku nebo spoj. Opět dojde k přechodu do stavu `necinny`.

Součástí třídy jsou ještě metody jako `Editor::pridatRezistor()` atd. pro přidání součástek do editoru. Jejich činnost je podobná, vždy se vytvoří nový objekt třídy `Soucastka`, přidá se do editoru a nastaví se nový mód - `vlozitSoucastku`.

6.4.3 Třída Soucastka

Třída dědí z `QGraphicsPathItem`. V konstruktoru třídy se vytvoří značka k příslušné součástce, jejíž typ je předáván jako parametr konstruktoru. Dále se nastaví vlastnosti jako možnosti vybrat součástku a pohybovat s ní. Nakonec se přidá se popisek součástky a inicializují se ostatní pomocné proměnné.

Každá součástka si uchovává v proměnné `int orientace` svojí aktuální orientaci: vodorovně nebo svisle. Se součástkou tedy může být i rotováno. Správné otočení součástky a jejího popisku zajišťuje metoda `Soucastka::rotate(qreal angle)`.

Kvůli vyššímu pohodlí uživatele byl tzv. `boundingRect` (ohraničující obdélník) rozšířen. Proto bylo nutné naimplementovat metodu `Soucastka::detekceVyberu(QPointF souradnice)`, která rozhoduje o výběru součástky při kliknutí do prostoru editoru.

6.4.4 Třída Spoj

Třída dědí z `QGraphicsLineItem`. Každý spoj je tvořen několika čárami. Jednotlivé čáry mohou být orientovány vodorovně nebo svisle. Čáry mají nastaveny jeden z možných úhlů: 0°, 90°, 180° nebo 270°. Spoj může být přichycen na součástce nebo na jiném spoji. Kolem každého spoje je naimplementován „koridor“, takže uživatel se nemusí přesně trefit myší na spoj, aby byl tento spoj vybrán. Se spojem je možno pohybovat.

6.4.5 Třída Graf

Třída dědí z `QGraphicsScene`. Osa x je logaritmická, osa y je lineární. Více charakteristik může být zobrazeno v jednom grafu. Průběhy jsou barevně odlišeny a uživatel může pomocí vyskakovacího menu zvolit barvu jinou. Vyskakovací menu je vyvoláno pomocí stisku pravého tlačítka na myši. V pravé části grafu je zobrazena legenda. Uživatel tedy vidí, jakou barvou je označena která simulace.

6.4.6 Třída Simulace

Třída dědí z `QObject`. Po vytvoření objektu je volána metoda `Simulace::nastaveniSimulace()`, která vytvoří dialogové okno pro zadání hodnot součástek, parametrů simulace a popis filtru diferenciálními rovnicemi. Dialogové okno je zobrazeno na obrázku 6.8.

Po stisku tlačítka „OK“ v dialogovém okně je spuštěna metoda `Simulace::spustitSimulaci()`. Jsou zkontrolovány zadané vstupní hodnoty a pokud dojde k chybě, tak je vypsáno chybové hlášení objektem `QMessageBox` a metoda se pomocí klíčového slova `return` opustí. Kontrolováno je:

- Zda bylo vyplněno číslo u rezistorů, kondenzátorů, cívek, zdrojů, počáteční frekvence a koncové frekvence.
- U rezistorů, kondenzátorů a cívek musí toto číslo být kladné.
- Dále musí platit, že počáteční frekvence je menší než koncová.
- Kontrola popisu modelu probíhá tak, že je nejprve zkompilován model a následně jsou možné chyby vypsány uživateli.

Pro nově spuštěnou simulaci je vytvořena na hlavním okně aplikace nová záložka s názvem „Simulace“, na které je zobrazen graf a výpis informací o provedených simulací. Objekt

graf je objektem třídy `Graf` a slouží k vykreslení amplitudové a fázové charakteristiky. Každá provedená simulace je číslována a uživatel si může vypsat hodnoty součástek, se kterými byla spuštěna, a dále si zvolit mezi zobrazením frekvenčních charakteristik. Následně jsou ve stejné metodě spuštěny další metody:

- `Simulace::vytvorDocasnyAdresar(QString aktualniAdresar, QString slozka)` = metoda vytvoří v adresáři s programem dočasný adresář, kam se poté uloží simulační model a kde proběhne jeho překlad.
- `Simulace::prevedDoSimlibu()` = metoda převede zápis diferenciálních rovnic v TKSL do SIMLIB. Simulační model je ve skutečnosti obyčejný program. Takže aby mohl být spuštěn, je nutné vytvořit funkci `main(int argc, char *argv[])`, kde se nastaví a spustí simulace. Jedná se o funkce pro řízení simulace uvedené v kapitole 6.2.3. Posledním krok je implementace funkce, která je pomocí objektu `Sampler` periodicky volána:

  ```
  void Sample1() {
      Expression f(FREKVENCE);
      filtr.u1 = ((AMPLITUDA*Sin(2*PI*f*T))).Value();
      Print(,,%g %g %g ‘‘, T.Value(), filtr.u1.Value(), filtr.u2.Value());
  }
  ```

  Na výstup každého simulačního běhu je uložen aktuální čas, napětí na vstupu a napětí na výstupu.
- `Simulace::vytvoritModel(QString pracovniAdresar)` = metoda uloží vytvořený model do pracovního adresáře.
- `Simulace::prelozitModel(QString pracovniAdresar)` = metoda zkompiluje simulační model v pracovním adresáři.
- `Simulace::spustitSimulacniModel()` = metoda spustí simulační model, zpracuje jeho výstup a výsledky uloží do grafu. Vyšetřování charakteristik probíhá stejným způsobem, jak bylo teoreticky uvedeno v kapitole 5.

  ```
  while(frekvence <= koncovaFrekvence){
          // spustim simulacni model
          castecnyVysledek = zahajitSimulaci(aktualniAdresar +
              QString(,,/simulace‘‘), frekvence, pocet_period);

          castecnyVysledekSplit = castecnyVysledek.split(,, ‘‘);
          castecnyVysledekSplit.removeLast();
  ```

  Cyklus běží, dokud neproměříme všechny body frekvenční charakteristiky. V každém cyklu je spuštěn model s aktuálně nastavenou frekvencí. Výsledky jsou uloženy do proměnné `castecnyVysledek`.

  ```
          // najdu ustaleny stav pro vystup
          ukoncit = false;
  ```

```
for (int k = castecnyVysledekSplit.size() - 1; k > 0; k -= 3){
    // najdu 1. nulu
    if (((QString) castecnyVysledekSplit[k]).toDouble() >= 0 &&
        ((QString) castecnyVysledekSplit[k - 3]).toDouble() <= 0){

        if (ukoncit == true){
            integral += ((QString) castecnyVysledekSplit[k]).toDouble();
            break;
        }
        integral = ((QString) castecnyVysledekSplit[k-3]).toDouble();
        k -= 3;
        ukoncit = true;
        continue;
    }

    // ,,integruji‘‘
    integral += ((QString) castecnyVysledekSplit[k]).toDouble();
}

// vyhodnotim, zda pokracovat ci premerit s vyssim poctem period
if (fabs(integral) > 0.0000001 /*presnost*/){
    if ((fabs(integral) > fabs(integral_predchozi)/2.0 ||
        pocet_period >= 10000) && integral_predchozi != 0){

        // nema smysl dal zvysovat frekvenci
        pocet_period = 10;
    }
    else{
        integral_predchozi = integral;
        pocet_period += 500;      // zvysim pocet merenych period
        continue;
    }
}

pocet_period = 10;
integral_predchozi = 0;
```

Aby bylo dosaženo odpovídající přesnosti, musíme hlídat, zda výstupní napětí je ustáleno. Výstupní napětí je ustáleno, pokud integrál přes jednu periodu jeho signálu je nulový. Pokud integrál není nulový, respektive jeho velikost je větší než zadaná přesnost, tak je simulace spuštěna znovu ale s vyšším počtem měřených period.

```
// najdu max hodnotu na vstupu, projdu jen 1 periodu
ukoncit = false;
maxVstup = ((QString)
    castecnyVysledekSplit[castecnyVysledekSplit.size() - 2]).toDouble();

for (int k = castecnyVysledekSplit.size() - 2; k > 0; k -= 3){
```

```
    if (((QString) castecnyVysledekSplit[k]).toDouble() > maxVstup){
        maxVstup = ((QString) castecnyVysledekSplit[k]).toDouble();
        maxCasVstup = ((QString) castecnyVysledekSplit[k-1]).toDouble();
        ukoncit = true;
    }
    else if (ukoncit == true){
        break;
    }
}
```

Hledám maximální hodnotu vstupního signálu. Hodnoty pro vstupní napětí jsou uloženy ve 2. sloupci, proto index přístupu je roven `castecnyVysledekSplit.size() - 2`. V cyklu `for` dekrementuji řídící proměnnou cyklu `k` o 3, protože simulace na výstup ukládá pro každý běh hodnoty `T.Value(), filtr.u1.Value(), filtr.u2.Value()`. Po načtení poslední nejvyšší hodnoty v rámci poslední periody cyklus opouštím.

```
// najdu max hodnotu na vystupu, projdu jen 1 periodu
zacit = false;
maxVystup = -1;
if (((QString)
    castecnyVysledekSplit[castecnyVysledekSplit.size() - 1]).toDouble()
    < 0){

    zacit = true;
}
for (int k = castecnyVysledekSplit.size() - 1; k > 0; k -= 3){
    if (((QString) castecnyVysledekSplit[k]).toDouble() >= 0 &&
        ((QString) castecnyVysledekSplit[k - 3]).toDouble() <= 0){

        if (zacit == false){
            // nasel jsem prvni ,,0‘‘
            zacit = true;
        }
        else{
            // nasel jsem druhou ,,0‘‘
            break;
        }
    }
    else if (zacit == false){
        continue;
    }

    if (((QString) castecnyVysledekSplit[k]).toDouble() > maxVystup){
        maxVystup = ((QString) castecnyVysledekSplit[k]).toDouble();
        maxCasVystup = ((QString) castecnyVysledekSplit[k-2]).toDouble();
    }
}
```

Hledám maximální hodnotu výstupního signálu. Hodnoty pro výstupní napětí jsou uloženy ve 3. sloupci, proto index přístupu je roven `castecnyVysledekSplit.size() - 1`. Po načtení poslední nejvyšší hodnoty v rámci poslední periody cyklus opouštím.

```
// -360/perioda * dt
faze = -360 * frekvence * (maxCasVystup - maxCasVstup);

// pridam body do seznamu
bodyProFrekChar.append(QPointF(frekvence, maxVystup/maxVstup));// prenos
bodyProFazChar.append(QPointF(frekvence, faze % 180));         // faze
```

Z odečtených hodnot lze vypočítat přenos a fázi.

```
if (frekvence == koncovaFrekvence)
    break;          // opustim cyklus

// zvysim frekvenci
if (frekvence + nastavNovouFrekvenci(frekvence) < koncovaFrekvence)
    frekvence += nastavNovouFrekvenci(frekvence);
else
    // provedu dokroceni na frekvenci pro konec simulace
    frekvence = koncovaFrekvence;

// aktualizuji status baru
((MainWindow *)this->parent())->nastavStatusBarText(
    frekvence/koncovaFrekvence * 100, true);

((MainWindow *)this->parent())->repaint();
```

V poslední části algoritmu se zvyšuje frekvence. Je nutné dát pozor na koncovou frekvenci, která nesmí být překročena. A dále se aktualizuje hodnota progress baru umístěného v status baru, aby uživatel měl přehled o průběhu výpočtu.

Kapitola 7

Hodnocení algoritmu řešení vytvořeného simulátoru

7.1 Časová náročnost výpočtu

Časová náročnost výpočtu závisí na rychlosti ustálení výstupního signálu. Pokud se integrací přes jednu periodu výstupního signálu zjistí, že výstupní napětí není ustáleno, je nutné spustit simulační model na dané frekvenci znovu. Pouštíme-li simulaci se stejnou frekvencí opakovaně, tak se prodlužuje doba výpočtu.

Ustálení výstupního signálu ovlivňuje mnoho faktorů jako parametry součástek, velikost vstupní frekvence. Experimentálně bylo ale zjištěno, že nejdéle trvá ustálení výstupního signálu v situaci, kdy přenos článku je malý (pohybuje se kolem nuly).

Nebyly nalezeny žádné výraznější časové komplikace.

7.2 Srovnání výsledků se světovými standardy

Provedlo se ověření správnosti výsledků v systému Matlab. Matlab provádí vyšetřování frekvenčních charakteristik odlišně. Na jeho vstup musí být zadána *přenosová funkce* filtru. Nikoliv popis obvodu pomocí diferenciálních rovnic. Výpočet Matlabu je tedy výrazně jednodušší.

Předpokládejme, že pomocí systému Matlab chceme vyšetřit charakteristiky dolnopropustného RC filtru, jehož odpor $R = 1\Omega$ a kapacita $C = 0,000159F$.

Časová konstanta obvodu τ se vypočítá jako:

$$\tau = RC = 1 \cdot 0,000159 = 0,000159s$$

Přenos filtru $G(s)$ je roven:

$$G(s) = \frac{1}{0,000159s + 1}$$

kde s je označení pro úhlovou frekvenci $[rad \cdot s^{-1}]$

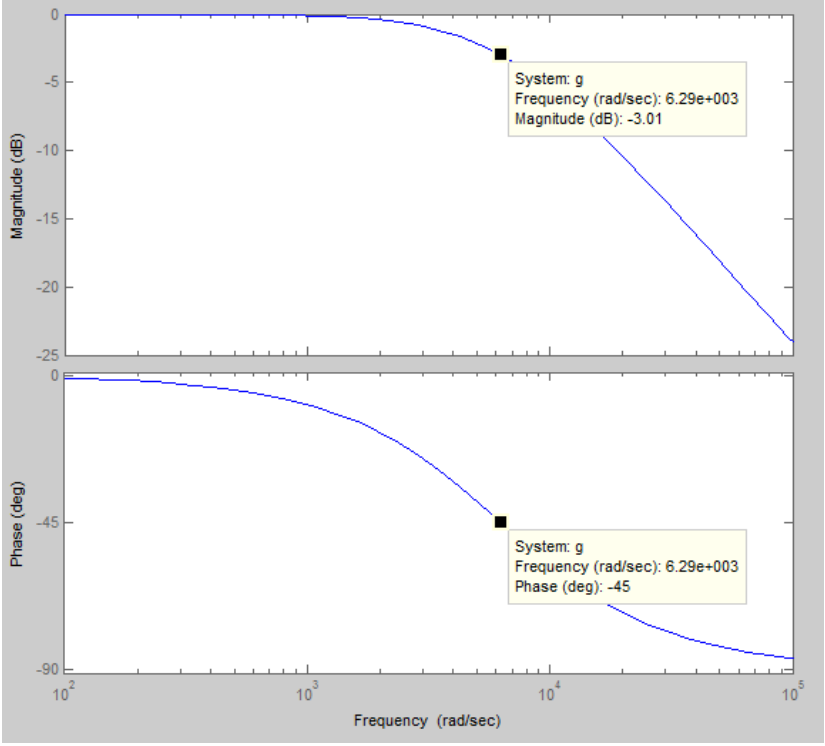
Pomocí následující posloupnosti příkazů v systému Matlabu se získají frekvenční charakteristiky na obrázku 7.1:

```
b=[1]
a=[0.000159 1]
g=tf(b,a)
bode(g)
```

K ověření výsledků nám pomůže hodnota mezního kmitočtu. Mezní kmitočet f_{mez} dolnopropustného RC filtru se podle [6] vypočítá ze vztahu

$$f_{mez} = \frac{1}{2\pi RC} \tag{7.1}$$

Po dosazení hodnot do vztahu 7.1, očekáváme mezní frekvenci $1000,97$Hz. Výsledky vrácené pomocí mého programu jsou na obrázcích 6.4 a 6.5 a odpovídají teoretickým předpokladům. Tedy, že filtr do frekvence kolem 1000Hz propouští signál na svůj výstup a také se shodují s výsledky vypočítanými pomocí systému Matlab. V případě Matlabu je potřeba dát pozor na jednotky, protože na ose x je uvedena *úhlová frekvence* a přenos je vypočítán v decibelech. Mezní frekvence se obvykle definuje jako pokles o 3dB. Této hodnotě dle obrázku 7.1 odpovídá úhlová frekvence $\omega = 6,29 \cdot 10^3$rad/s. Přepočítáme-li úhlovou frekvenci na frekvenci v Hz jako $f = \frac{\omega}{2\pi}$, vyjde nám hodnota $1001,08$Hz. Což po drobném zaokrouhlení souhlasí s dříve vypočítanou mezní frekvenci $1000,97$Hz. Obdobně je nutné přepočítat i přenos uvedený v decibelech. Z obrázku 6.4 je vidět, že pro mezní frekvenci můj program vypočítal přenos jako $0,7$. Po dosazení této hodnoty do vztahu 5.2, nám výjde $-3,01$. Což opět odpovídá výsledku vraceného pomocí Matlabu.

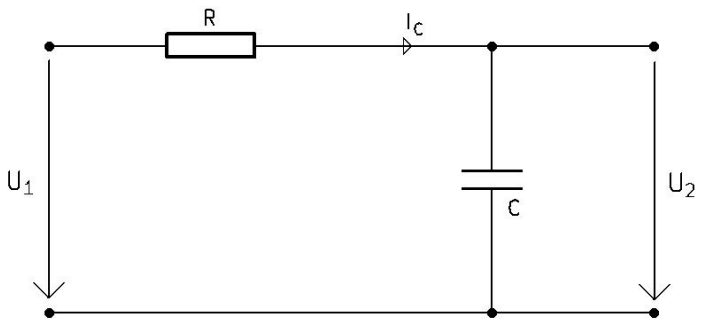


Obrázek 7.1: Charakteristiky filtru pomocí systému Matlab
Zdroj: Vlastní

Kapitola 8

Příklady pro vyšetřování frekvenčních charakteristik obvodů R, L, C

8.1 Příklad 1



Obrázek 8.1: Integrační článek - dolní propust
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.1 jsou:

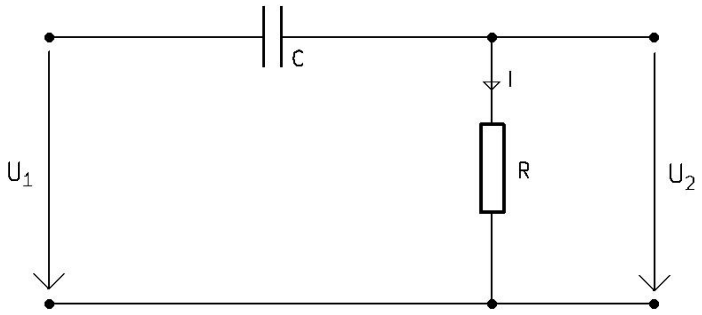
$$\begin{aligned} i_C &= \frac{u_1 - u_2}{R} \\ u_2' &= \frac{1}{C}\frac{u_1 - u_2}{R} \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct INTEG_CLANEK{

Variable U1;
```

```
Parameter R, C;
Integrator U2;

INTEG_CLANEK( double R, double C ):

R(R), C(C),

U2(1/C * (U1-U2)/R) {}
};
```

8.2 Příklad 2



Obrázek 8.2: Derivační článek - horní propust
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.2 jsou:

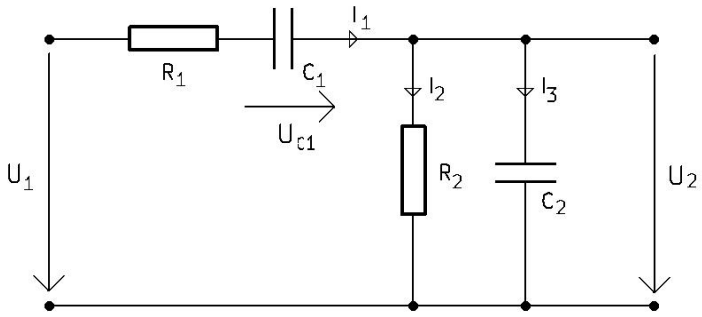
$$\begin{aligned} i &= \frac{u_2}{R} \\ i &= C(u_1 - u_2)' \Rightarrow \\ u_2 &= u_1 - \int \frac{i}{C} \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct DERIVACNI_CLANEK{

Variable U1;
Parameter R, C;
Expression I, U2;
Integrator UC;

DERIVACNI_CLANEK( double R, double C ):
```

```
R(R), C(C),

I(U2/R),

UC(I/C),

U2(U1 - UC) {}
};
```

8.3 Příklad 3



Obrázek 8.3: Wienův článek - pásmová propust
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.3 jsou:

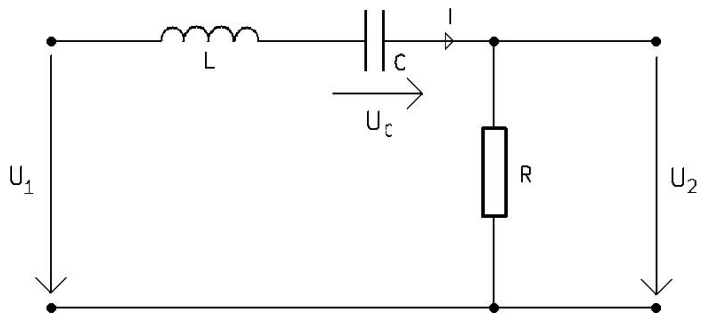
$$\begin{aligned} i_1 &= \frac{u_1 - u_{C1} - u_2}{R_1} \\ i_2 &= \frac{u_1 - u_{C1} - i_1 R_1}{R_2} \\ i_3 &= i_1 - i_2 \\ u'_{C1} &= \frac{1}{C_1} i_1 \\ u'_2 &= \frac{1}{C_2} i_3 \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct WIENUV_CLANEK{

Variable U1;
Parameter R1, R2, C1, C2;
Expression i1, i2, i3;
Integrator U2, uC1;
```

```
WIENUV_CLANEK( double R1, double R2, double C1, double C2 ):

R1(R1), R2(R2), C1(C1), C2(C2),

i1((U1 - uC1 - U2)/R1),
i2((U1 - uC1 - i1*R1)/R2),
i3(i1 - i2),

uC1(1/C1*i1),
U2(1/C2*i3) {}
};
```

8.4 Příklad 4



Obrázek 8.4: Rezonanční pásmová propust
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.4 jsou:

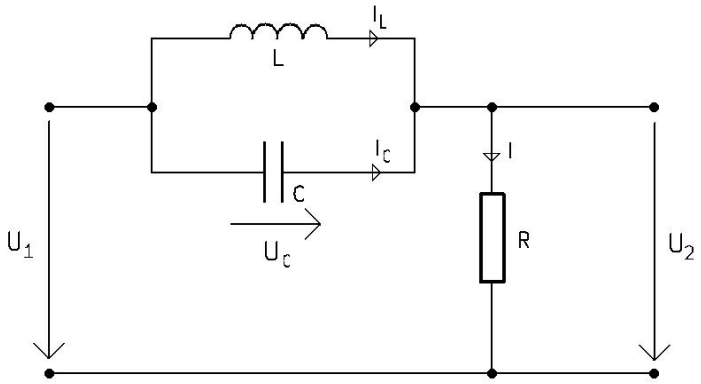
$$\begin{aligned} i' &= \frac{1}{L}(u_1 - u_C - u_2) \\ u'_C &= \frac{1}{C} i \\ u_2 &= iR \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct PASMOVA_PROPUST{

Variable U1;
Parameter R, C, L;
Expression U2;
Integrator UC, I;
```

```
PASMOVA_PROPUST( double R, double C, double L ):

R(R), C(C), L(L),

I(1/L * (U1 - UC - U2)),

UC(1/C*I),
U2(I*R) {}
};
```

8.5 Příklad 5



Obrázek 8.5: Rezonanční pásmová zádrž
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.5 jsou:

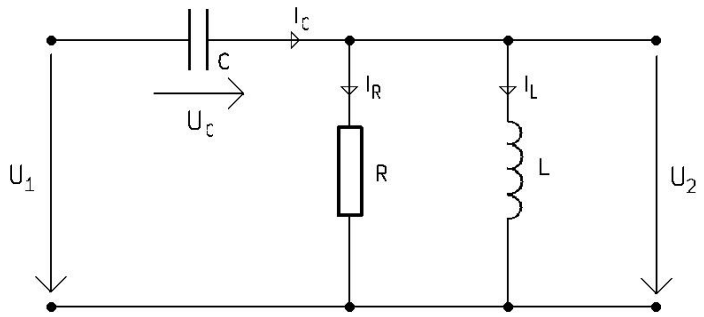
$$\begin{aligned} i &= \frac{u_1 - u_C}{R} \\ i'_L &= \frac{1}{L}(u_1 - u_2) \\ i_C &= i - i_L \\ u'_C &= \frac{1}{C} i_C \\ u_2 &= iR \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct PASMOVA_ZADRZ{

Variable U1;
Parameter R, C, L;
Expression U2, i, iC;
```

```
Integrator uC, iL;

PASMOVA_ZADRZ( double R, double C, double L ):

R(R), C(C), L(L),

i((U1 - uC)/R),
iL(1/L * (U1 - U2)),
iC(i - iL),

uC(1/C*iC),
U2(i*R) {}
};
```

8.6 Příklad 6



Obrázek 8.6: Sério-paralelní RLC článek
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.6 jsou:

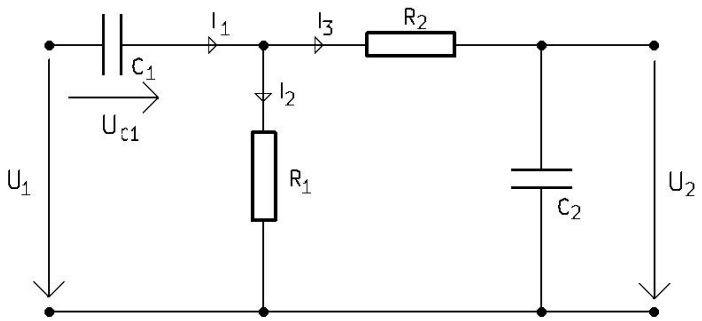
$$\begin{aligned} i_C &= i_R + i_L \\ i_R &= \frac{u_1 - u_C}{R} \\ i'_L &= \frac{1}{L}(u_1 - u_C) \\ u'_C &= \frac{1}{C} i_C \\ u_2 &= R i_R \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct SP_RLC_CLANEK{

Variable U1;
```

```
Parameter R, C, L;
Expression iC, iR, U2;
Integrator iL, uC;

SP_RLC_CLANEK( double R, double C, double L ):

R(R), C(C), L(L),

iC(iR+iL),
iR((U1 - uC)/R),
iL(1/L*(U1 - uC)),

uC(1/C*iC),
U2(R*iR) {}
};
```

8.7 Příklad 7



Obrázek 8.7: Pásmová propust
Zdroj: Vlastní

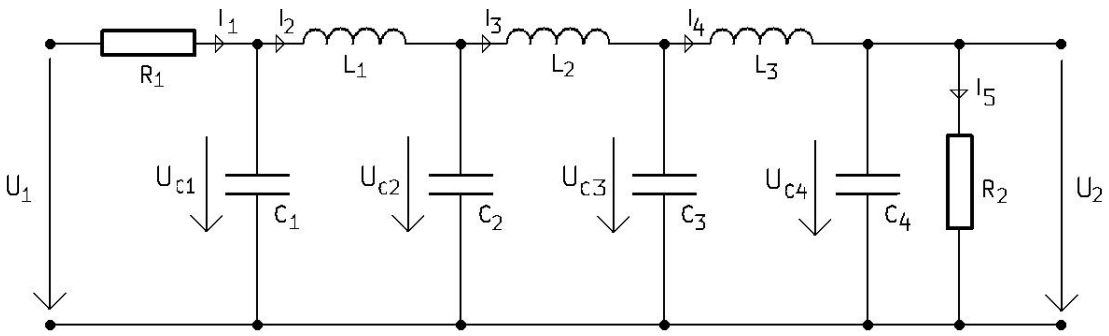
Diferenciální rovnice popisující obvod na obrázku 8.7 jsou:

$$\begin{aligned} i_1 &= i_2 + i_3 \\ i_2 &= \frac{u_1 - u_{C1}}{R_1} \\ i_3 &= \frac{u_1 - u_{C1} - u_2}{R_2} \\ u'_{C1} &= \frac{1}{C_1} i_1 \\ u'_2 &= \frac{1}{C_2} i_3 \end{aligned}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct PASMOVA_PROPUST{

Variable U1;
Parameter R1, R2, C1, C2;
Expression i1, i2, i3;
Integrator U2, uC1;

PASMOVA_PROPUST( double R1, double R2, double C1, double C2 ):

R1(R1), R2(R2), C1(C1), C2(C2),

i1(i2+i3),
i2((U1 - uC1)/R1),
i3((U1 - uC1 - U2)/R2),

uC1(1/C1*i1),
U2(1/C2*i3) {}
};
```

8.8 Příklad 8



Obrázek 8.8: Čebyševův filtr
Zdroj: Vlastní

Diferenciální rovnice popisující obvod na obrázku 8.8 jsou:

$$\begin{array}{ll} i_1 = \frac{u_1 - u_{C1}}{R_1} & u'_{C1} = \frac{1}{C_1}(i_1 - i_2) \\ i'_2 = \frac{1}{L_1}(u_{C1} - u_{C2}) & u'_{C2} = \frac{1}{C_2}(i_2 - i_3) \\ i'_3 = \frac{1}{L_2}(u_{C2} - u_{C3}) & u'_{C3} = \frac{1}{C_3}(i_3 - i_4) \\ i'_4 = \frac{1}{L_3}(u_{C3} - u_{C4}) & u'_{C4} = \frac{1}{C_4}(i_4 - i_5) \\ i_5 = \frac{u_{C4}}{R_2} & u_2 = R_2 i_5 \end{array}$$

Popis modelu v jazyce C++ s využitím knihovny SIMLIB:

```
struct CEBYSEVUV_FILTR{
```

```
Variable U1;
Parameter R1, R2, C1, C2, C3, C4, L1, L2, L3;
Expression i1, i5, U2;
Integrator i2, i3, i4, uC1, uC2, uC3, uC4;

CEBYSEVUV_FILTR( double R1, double R2, double C1, double C2, double C3,
double C4, double L1, double L2, double L3 ):

R1(R1), R2(R2), C1(C1), C2(C2), C3(C3), C4(C4), L1(L1), L2(L2), L3(L3),

i1((U1 - uC1)/R1),
i2(1/L1*(uC1-uC2)),
i3(1/L2*(uC2-uC3)),
i4(1/L3*(uC3-uC4)),
i5(uC4/R2),

uC1(1/C1*(i1-i2)),
uC2(1/C2*(i2-i3)),
uC3(1/C3*(i3-i4)),
uC4(1/C4*(i4-i5)),
U2(R2*i5) {}
};
```

Kapitola 9

Závěr

Při návrhu uživatelského rozhraní pro schématické zadávání elektronických obvodů jsem se inspiroval profesionálními programy. Práce s editorem je uživatelsky přívětivá. Součástky jsou automaticky přitahovány k mřížce. Rovněž práce se spoji a součástkami nevyžaduje příliš grafické přesnosti od uživatele.

Hlavním cílem práce byla implementace simulátoru vyšetřující amplitudovou a fázovou charakteristiku příslušného obvodu. Tento proces je plně automatizovaný. Délka výpočtu závisí na rozsahu frekvencí zadané uživatelem a na rychlosti ustálení výstupního signálu. Kontrolovat stav ustálení výstupního signálu je velice důležité, jinak bychom se mohli dopouštět nepřesnosti.

Program je napsán v jazyku C++ s využitím knihoven Qt 4.7.0 a SIMLIB 3.01. Program byl vyvíjen pod systémem Linux. S programem bylo provedeno množství testů. Byly vyšetřeny charakteristiky všech obvodů uvedených v kapitole 8 a výsledky srovnány se systémem Matlab.

Možným rozšířením funkcí simulátoru by byla implementace třídy zajišťující automatické sestavení diferenciálních rovnic používaných pro popis elektronického obvodu. V současnosti jsou rovnice zadávány v syntaxi systému TKSL. Hlavně díky jeho přehlednosti, jednoduchosti a blízkosti k matematickému zápisu.

Výsledky práce mohou být využity při výuce v předmětech ITO a IPR.

Literatura

[1] BLAHOVEC, A.: *Elektrotechnika II.* Praha: Informatorium, Čtvrté vydání, 2003, ISBN 80-7333-013-X.

[2] BLANCHETTE, J.; SUMMERFIELD, M.: *C++ GUI programming with Qt 4.* Courier in Stoughton, Massachusetts, první vydání, 2006, ISBN 0-13-187249-4.

[3] DOSTÁL, T.; AXMAN, V.: Elektrické filtry. *Vysoké učení technické VUT*, 2004.

[4] FAJMON, B.; RŮŽIČKOVÁ, I.: Matematika 3. *Vysoké učení technické VUT*, 2005.

[5] KUNOVSKÝ, J.: High Performance Computing [online]. `http://www.fit.vutbr.cz/~kunovsky/TKSL`, 2005-03-17 [cit. 2011-04-05].

[6] LÁNÍČEK, R.: *Elektronika: Obvody, součástky, děje.* Praha: Nakladatelství BEN - technická literatura, první vydání, 1998, ISBN 80-86056-25-2.

[7] PERINGER, P.: Modelování a simulace. *Vysoké učení technické VUT*, 2008.

[8] PERINGER, P.: Popis simulační knihovny SIMLIB [online]. `http://www.fit.vutbr.cz/~peringer/SIMLIB/doc/html-cz`, 2011-03-13 [cit. 2011-03-13].

[9] Qt: Qt - Cross-platform application and UI framework [online]. `http://qt.nokia.com/`, 2011-03-23 [cit. 2011-04-06].

[10] URBÁNEK, R.: *Frekvenční charakteristiky.* Diplomová práce, FIT VUT v Brně, Brno, 2006.

Dodatek A

Metriky kódu

Počet souborů:	13 souborů
Počet řádků zdrojového textu:	5 375 řádků
Velikost spustitelného souboru:	2 155 121B (systém Linux, 32 bitová architektura, při překladu bez ladících informací)

Dodatek B

Hlavičkové soubory naimplementovaných tříd

B.1 Třída MainWindow

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QtGui>

#include ,,editor.h‘‘
#include ,,soucastka.h‘‘
#include ,,simulace.h‘‘


class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow();
    ~MainWindow();

    void nastavStatusBarText(int hodnota, bool viditelnost);

    QAction *odstranitSoucatkuNeboSpojAkce;
    QAction *otocitSoucastkuPoSmeruAkce;
    QAction *otocitSoucastkuProtiSmeruAkce;
    QAction *pridatSpojAkce;
    QAction *nastavitKurzorAkce;

    QTabWidget *zalozkaObvody;

    Editor *scena;
```

```
    QGraphicsView *view;

    Simulace *simulace;

public slots:
    void pridatRezistor();
    void pridatKondenzator();
    void pridatCivku();
    void pridatZdroj();
    void pridatOsc();

    void nastavitKurzor();
    void pridatSpoj();

    void odstranitSoucatkuNeboSpoj();

    void otocitSoucastkuPoSmeru();
    void otocitSoucastkuProtiSmeru();

    void spustitSimulaci();
    void ukoncitSimulaci();

    void zavrit();
    void novyProjekt();
    void zobrazOProgramu();

    bool eventFilter(QObject* obj, QEvent* event);

protected:
    void closeEvent(QCloseEvent *e);

private:
    void vytvorAkce();
    void vytvorMenu();
    void vytvorToolbar();
    void vytvorPracovniPlochu();

    void odstranitZeSeznamu(Soucastka* soucastka);

    QHBoxLayout *rozlozeni;

    QMenu *souborMenu;
    QMenu *napovedaMenu;
    QMenu *vlozitMenu;
    QMenu *upravitMenu;
    QMenu *simulaceMenu;

    QAction *novyProjektAkce;
```

```
    QAction *konecAkce;
    QAction *oProgramuAkce;

    QToolBar *soucastkyToolBar;
    QToolBar *ovladaniToolBar;
    QToolBar *simulaceToolBar;

    QAction *pridatRezistorAkce;
    QAction *pridatKondenzatorAkce;
    QAction *pridatCivkaAkce;
    QAction *pridatZdrojAkce;
    QAction *pridatOscAkce;

    QAction *spustitSimulaciAkce;
    QAction *ukoncitSimulaciAkce;

    QProgressBar *status;
};

#endif // MAINWINDOW_H
```

B.2 Třída Editor

```
#ifndef EDITOR_H
#define EDITOR_H

#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include <QDebug>
#include <QMenuBar>

#include ,,soucastka.h‘‘
#include ,,spoj.h‘‘

class QGraphicsSceneMouseEvent;
class QMenu;
class QPointF;
class QGraphicsLineItem;
class QFont;
class QGraphicsTextItem;
class QColor;

class Spoj;
class Soucastka;
class MainWindow;

class Editor : public QGraphicsScene
{
```

```
    Q_OBJECT

public:
    enum Mod { necinny /*0*/, vlozitSoucastku /*1*/, vlozitSpoj /*2*/,
               vlozitSpojPokr/*3*/, ukoncitSpoj /*4*/,
               pohybSoucastka /*5*/, pohybSpoj /*6*/ };

    Editor(MainWindow *parent);
    ~Editor();

    void uvolniSeznamySoucastek();

    Spoj* addSpoj(const QLineF & line, Spoj *predchoziSpoj);

    void vykresliMrizku(qreal sirka, qreal vyska);

    void nastavMod(Mod mod);
    int vratMod(){ return aktualniMod; }

    void pridatRezistor();
    void pridatKondenzator();
    void pridatCivku();
    void pridatZdroj();
    void pridatOsc();

    QList <Soucastka *> seznamRez, seznamKon, seznamCivek, seznamZdroju, seznamOsc;

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent);
    void mouseMoveEvent(QGraphicsSceneMouseEvent *mouseEvent);
    void mouseReleaseEvent(QGraphicsSceneMouseEvent *mouseEvent);
    void wheelEvent(QGraphicsSceneWheelEvent *wheelEvent);

    void popupMenu(QGraphicsSceneMouseEvent* event);

private:
    int vratNoveOznaceniProSoucastku(int typSoucastky);

    Spoj* spojAt(QPointF souradniceMysi);
    Spoj* spojAtKonec(QPointF souradniceMysi);
    Soucastka* soucastkaAt(QPointF souradniceMysi);

    QPen peroMrizka;

    int aktualniMod;

    Soucastka *soucastka;
    Spoj *spoj, *spojPredchozi;
```

```
    Spoj *vybranySpoj;

    QGraphicsLineItem *cara;
    qreal uhel;
    QPointF souradnice;
};

#endif // EDITOR_H
```

B.3 Třída Soucastka

```
#ifndef SOUCASTKA_H
#define SOUCASTKA_H

#include <QGraphicsPathItem>

#include ,,spoj.h‘‘

class Spoj;


class Soucastka : public QGraphicsPathItem
{
public:
    enum Typ { Rezistor, Kondenzator, Civka, Zdroj, Osciloskop };
    enum Orientace { Vodorovne, Svisle };
    enum { Type = UserType + 1 };       // typ

    Soucastka(int typ, QString popisek, QGraphicsScene *rodic);
    ~Soucastka();

    int type() const{
        return Type;
    }

    void setPos(qreal x, qreal y);
    void rotate(qreal angle);

    void nastavitOrientaci(int orientacePar);
    int vratOrientaci();

    int vratTyp();

    QString vratPopisek();
    QGraphicsTextItem *text;

    void prichytitNaMrizku(QPointF souradniceMysi, int velikostMrizky);
```

```
    void upravitPoziceSpoju();
    void upravitPoziceSpojuVlevo();
    void upravitPoziceSpojuVpravo();

    bool detekceVyberu(QPointF souradnice);

    QRectF boundingRect() const;
    QRectF namapujBoundingRectDoSceny();

    Spoj *spojZleva;
    Spoj *spojZprava;

private:
    QPainterPath vratZnackuZdroje();
    QPainterPath vratZnackuOsciloskopu();

    int orientace;
    int typSoucastky;
};

#endif // SOUCASTKA_H
```

B.4 Třída Spoj

```
#ifndef SPOJ_H
#define SPOJ_H

#include <QGraphicsLineItem>
#include <QGraphicsRectItem>
#include <QDebug>

#include ,,soucastka.h‘‘
#include ,,editor.h‘‘

class Soucastka;
class Editor;

class Spoj : public QGraphicsLineItem
{
public:
    enum { Type = UserType + 2 };           // typ
    enum Orientace { Vodorovne, Svisle };
    enum spojNaSoucastce { Zleva, Zprava };

    Spoj(const QLineF &line, Editor *rodic);
    ~Spoj();
```

```
    int type() const { return Type;  }

    bool detekceVyberu(QPointF souradnice);

    void mouseMoveEvent(QPointF souradnice, Soucastka *soucastka = 0);

    void nastavPredchoziSpoj(Spoj *predchozispoj);
    void nastavNasledujiciSpoj(Spoj *nasledujicispoj);

    int prichytitNaSoucastku(Soucastka *soucastka, QPointF souradniceMysi);
    void prichytitNaSpoj(Spoj *spoj, QPointF souradniceMysi);

    qreal vratUhel();
    qreal vratUhelProNovouCaru(QPointF souradnice);

    void upravUzly();

    Soucastka *soucastkaZleva;
    Soucastka *soucastkaZprava;

    Spoj *spojZleva;
    Spoj *spojZprava;
    QGraphicsEllipseItem *uzelZleva;
    QGraphicsEllipseItem *uzelZprava;

    QList<Spoj *> seznamPrichycenychSpojuP1;
    QList<Spoj *> seznamPrichycenychSpojuP2;

protected:
    void vyberSvojeSousedy(bool vybrat);
    bool spojPrichycenNaSpoji(Spoj *spoj);

private:
    QRectF boundingRect() const;

    void upravSousedniSoucastky(Soucastka *soucastka);
    void upravSousedniSpoje();
    void upravPrichyceneSpoje();

    int prichytitNaSoucastkuStart(QPointF souradniceMysi, Soucastka *soucastka);
    int prichytitNaSoucastkuPokracovani(QPointF souradniceMysi, Soucastka *soucastka);

    void prichytitNaSpojStart(QPointF souradniceMysi, Spoj *spoj);
    void prichytitNaSpojPokracovani(QPointF souradniceMysi, Spoj *spoj);

    bool spojPrichycenNaSoucastce(Soucastka *soucastka);

    void upravOdkazyNaSousedy();
```

```
    Spoj *predchoziSpoj;
    Spoj *nasledujiciSpoj;

    qreal uhel;
    Editor *scenaRodic;
};

#endif // SPOJ_H
```

B.5 Třída Graf

```
#ifndef GRAF_H
#define GRAF_H

#include <QGraphicsScene>
#include <QDebug>
#include <QGraphicsTextItem>
#include <QColorDialog>
#include <QMenu>

#include <QGraphicsSceneMouseEvent>
#include <math.h>

class Graf : public QGraphicsScene
{
    Q_OBJECT

public:
    Graf();
    ~Graf();

    enum Zobrazit { FREK_CHAR, FAZ_CHAR };

    void pridatBodyProFrekChar(int cisloSimulace, QList<QPointF> body);
    void pridatBodyProFazChar(int cisloSimulace, QList<QPointF> body);

    void zobrazFrekChar();
    void zobrazFazChar();

    void pridatKrivku(int cisloSimulace);
    void odebratKrivku(int cisloSimulace, int odkud);

    void nastavHranice(QSize velikost);

protected:
    void mousePressEvent(QGraphicsSceneMouseEvent *mouseEvent);
```

```
    void popupMenu(QGraphicsSceneMouseEvent* event);

private slots:
    void zmenitBarvu(int cisloSimulace);
    void slotEmitujiciMujSignal();

signals:
    void mujSignal(int cisloSimulace);

private:
    void vykresliLogOsuX(QPointF startBod, QPointF konecBod);
    void vykresliLinOsuY(int cisloSimulace);
    void vykresliKrivku(int cisloSimulace);
    void vykresliLegendu();

    QColor vygenerujBarvu();

    void odstranitVsechnoZeSceny();

    qreal prevedDoIntervaluZacatek(qreal bodNaOseX);

    qreal prepocitatNaLog(qreal hodnota);
    qreal prepocitatNaLin(qreal hodnota);

    double vratKrokProNacarkovaniOsy(double krok);
    bool doublePorovnani(double op1, double op2);
    bool doubleMensi(double op1, double op2, bool orequal = false);

    int stav;

    qreal delkaDekady;
    qreal pocatecniBod;
    qreal koncovyBod;
    qreal polohaVyskaOsyX;
    qreal maxHodnotaY;

    QGraphicsLineItem *osaX, *osaY;

    QSize dolniPravyRoh;

    // aktualne zobrazovane krivky a jim prislusne cisla simulaci
    QList<int> cislaSimulaci;
    QList< QList<QGraphicsLineItem *> > seznamKrivek;  // obsahuje bodyProFrekChar
                                                       // nebo bodyProFazChar
    QList<int> seznamKrivekCislaSim;
    QList<QGraphicsTextItem *> legenda;
    QList<QGraphicsRectItem *> legendaCtverce;
    QList<int> barvaCislaSim;
```

```
    QList<QColor> barva;

    // body pro vsechny simulace
    QList< QList<QPointF> > bodyProFrekChar;
    QList< QList<QPointF> > bodyProFazChar;

    QAction *zmenitBarvuAkce;
};

#endif // GRAF_H
```

B.6 Třída Simulace

```
#ifndef SIMULACE_H
#define SIMULACE_H

#include <QMainWindow>
#include <QtGui>
#include <QApplication>
#include <QProcess>
#include <QStringList>

#include ,,graf.h‘‘

class Simulace: public QObject
{
    Q_OBJECT

public:
    Simulace(QMainWindow *parent = 0);
    ~Simulace();

    void nastaveniSimulace();

public slots:
    void spustitSimulaci();
    void nespoustetSimulaci();

    void kliknutoNaVypis(QTreeWidgetItem *item, int column);

private:
    void spustitSimulacniModel();
    double nastavNovouFrekvenci(double frekvence);

    void popisekSimulace();
    QString prevedDoSimlibu();
    QString vytvorDeklaraciProm();
    QString vytvorNapojeniParametru();
```

```
    QString vytvorRovnice();
    QString hodnotySoucastek();
    QString vytvorSample();
    QString vytvorMain();

    template<class T> void uvolniPamet(T &list);

    QString vratPracovniAdresar();
    void vytvorDocasnyAdresar(QString aktualniAdresar, QString slozka);
    void odstranitDocasnyAdresar(QString aktualniAdresar, QString slozkaOdstranit);
    void vytvoritModel(QString pracovniAdresar);
    bool prelozitModel(QString pracovniAdresar);
    QString zahajitSimulaci(QString pracovniAdresar, double frekvence,
                            int pocet_period);

    bool zkontrolujVstupniHodnoty();
    bool jeCislo(QString str, int rozsah);
    void vypisChybu(QString hlaseni);

    QString ocislovatRadky(QString text);

    QMainWindow *oknoSimulace;
    QMainWindow *dialogoveOkno;
    QTextEdit *popisModelu;
    QLineEdit *startSim, *konecSim;

    Graf *graf;
    QGraphicsView *view;

    // vypis
    QTreeWidget *vypis;
    QList <QTreeWidgetItem *> simulace;

    bool novaSimulace;
    QString aktualniAdresar;
    int cisloSimulace;
    QString popisModeluSimlib;

    QList <QLineEdit *> seznamRez, seznamKon, seznamCivek, seznamZdroju;

    // stavy konecneho automatu
    enum Stav { ZACATEK, PROMENNA, PROMENNA_2, INTEGRATOR, POCATECNIPODMINKA,
                VYRAZ, CHYBA };

    enum Cisla { KLADNA, ZAPORNA, VSECHNA };
};

#endif // SIMULACE_H
```